

**IMAGE PROCESSING PIPELINE BASED ON
COUPLED OSCILLATOR MODELS**

by

John Carpenter

B.S. in Computer Engineering, University of Pittsburgh, 2011

Submitted to the Graduate Faculty of the
Kenneth P. Dietrich School of Arts and Sciences in partial fulfillment
of the requirements for the degree of
M.S. in Computer Engineering

University of Pittsburgh

2014

UNIVERSITY OF PITTSBURGH
DIETRICH SCHOOL OF ARTS AND SCIENCES

This thesis was presented

by

John Carpenter

It was defended on

December 11, 2013

and approved by

Dr. Donald M. Chiarulli, PhD, Professor, Department of Computer Science

Dr. Steven P. Levitan, PhD, Professor, Department of Electrical and Computer Engineering

Dr. Jun Yang, PhD, Professor, Department of Electrical and Computer Engineering

Dr. Amro A. El-Jaroudi, PhD, Professor, Department of Electrical and Computer Engineering

Copyright © by John Carpenter

2014

IMAGE PROCESSING PIPELINE BASED ON COUPLED OSCILLATOR MODELS

John Carpenter, M.S.

University of Pittsburgh, 2014

For decades, researchers have been developing algorithms for image processing pipelines. Image Processing Pipelines (IPPs) are algorithmic constructions built to iteratively modify an input image into a series of abstractions for the purposes of decoding its contents into a higher level representation. There have been many proposed IPPs, varying in both physical construction, and in algorithmic paradigm, but by and large these propositions have been based in Boolean computation and arithmetic. Studies and trends have shown that Boolean computers are hitting a theoretical ceiling on their performance in terms of transistor size, energy consumption/heat dissipation, clock rates, and by extension computational time. Due to these issues, researchers have proposed using non-Boolean approaches, where possible, for various computations in common algorithms. One of the emerging technologies in the field of non-Boolean computation has been the use of coupled oscillators. A proposed use of coupled oscillators is for pattern matching, which can also be interpreted as a high-dimensional distance measurement. Using an approach based on the use of coupled oscillators as a basic computational primitive, this work aims to utilize the benefits gained from this new computational paradigm to gain performance in terms of both speed and power with respect to IPPs, without decreasing the accuracy of their algorithms.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	X
1.0 INTRODUCTION.....	1
1.1 MOTIVATION	2
1.2 PROBLEM STATEMENT	4
1.3 THESIS ORGANIZATION.....	6
2.0 BACKGROUND AND RELATED WORK	7
2.1 IMAGE PROCESSING BACKGROUND	7
2.2 PREVIOUS WORK ON IMAGE PROCESSING.....	9
2.2.1 Frontend Previous Work	9
2.2.2 HMAX.....	15
2.2.3 Convolutional Neural Networks.....	18
2.2.4 Feature Vector Classifiers.....	20
2.2.5 Training Methodologies	24
2.3 PREVIOUS WORK ON IPP HARDWARE ACCELERATORS.....	26
2.4 NON-BOOLEAN COMPUTATION WITH OSCILLATORS.....	27
2.4.1 Empirical DoM Model.....	28
3.0 USING GRBF AS AN OSCILLATOR DOM MODEL	30
4.0 HMAX	33

4.1.1	Data Set.....	33
4.2	BASILINE IMPLEMENTATION.....	34
4.2.1	HMAX Parameters.....	34
4.3	USING OSCILLATOR MODELS.....	40
4.4	S1 LAYER MODIFICATION.....	41
4.5	S2 LAYER MODIFICATION.....	46
4.6	CLASSIFIER MODIFICATION.....	48
4.7	OVERALL PERFORMANCE AND RESULTS.....	50
5.0	HMAX MODIFICATION DISCUSSION.....	52
5.1.1	Parameterization of Speedup.....	52
5.1.2	Discussion of Power Savings.....	61
6.0	CONCLUSION, SUMMARY, AND FUTURE WORK.....	63
6.1	SUMMARY.....	63
6.2	CONCLUSIONS.....	64
6.3	FUTURE WORK.....	65
7.0	BIBLIOGRAPHY.....	66

LIST OF TABLES

Table 1 – Example images from Stanford Tower Dataset used as input to HMAX IPP	33
Table 2 – Examples of Gabor Filtering output on image samples from the 5 classes using multiplicative convolution	37
Table 3 – Benchmark IPP Parameters and Classification Percentage Performance.....	39
Table 4 – Examples of Gabor Filtering output on samples images from the 5 classes using DoM convolution	45
Table 5 – Summary of oscillator model and parameter modifications to the HMAX IPP, along with the results from the benchmark IPP. A * denotes a change in the parameters of the benchmark IPPTable 3	51
Table 6 – Number of computations which are the main computational differences between the benchmark and modified IPP. These are used as the basis for determining the computational reduction of the modified IPP. The superscripts are pointers to the list below to show how to compute the value shown in the table.	54
Table 7 – CR values for three examples of D/M and k parameter pairings.....	60
Table 8 – Summary of the results in this work. Each column represents a location in the algorithm for a modification to the benchmark IPP, and each row specifies a result associated with that modification.	61

LIST OF FIGURES

Figure 1 – Sample image from Stanford Tower Datasets with bounding boxes around objects of interest.....	8
Figure 2 - © 1998 IEEE - general architecture of Itti’s saliency model (Itti, Kock, & Niebur, 1998)	11
Figure 3 - ©1998 IEEE– Example of the operation with natural image. First, three saliency channels are summed to create total saliency, and areas are chosen, and then inhibited in cyclical fashion to determine highly saliency regions of interest [20].....	12
Figure 4 – Data flow of the HMAX-based IPP.....	15
Figure 5 – ©2006 IEEE - Pictorial representation of HMAX [24]	16
Figure 6 – Sigmoid function output shape	20
Figure 7 – SVM classifier versus nearest neighbor classifier: classifies the test input as class B in 2D space.....	22
Figure 8 – Neural Network classifier with 5 features and 3 output classes	23
Figure 9 – Example of how to modify input image to perform a full convolution operation (\bullet_f)	25
Figure 10 – GRBF with a $\gamma=1$	31
Figure 11 – Scatter plot demonstrating the linear relationship between DoM and GRBF for input vectors of size 16, 32, and 64.....	32

Figure 12 - Number of Scales output by the Image Layer versus Classification Percentage	35
Figure 13 – classification percentages versus S2 template size for the benchmark HMAX IPP .	38
Figure 14 – Sweep of variables γ and Gabor filter size using DoM oscillator model as a replacement for dot product in the convolution in the S1 layer to optimize classification percentage	42
Figure 15 – Sweep of variables γ and Gabor filter size using GRBF oscillator model as a replacement for dot product in the convolution in the S1 layer to optimize classification percentage	42
Figure 16 – Second sweep of variables γ and Gabor filter size using GRBF oscillator model as a replacement for dot product in the convolution in the S1 layer to optimize classification percentage	44
Figure 17 – S2 Layer DoM template response, γ parameter's impact on the classification performance	47
Figure 18 – S2 Layer GRBF template response, γ parameter's impact on the classification performance	48
Figure 19 – Classifier DoM template response, γ parameter's impact on the classification performance	49
Figure 20– Classifier GRBF template response, γ parameter's impact on the classification performance	50
Figure 21 - Plot of the computational reduction of (top left, top right, bottom left, bottom right) S1 layer, S2 layer, classifier, and entire IPP backend as a function of time to compute DoM (D), time to compute a multiplication in CMOS (M), and number of coupled oscillators in the DoM circuit (k). The black plane in each graph represents a computational reduction of 1.	59

ACKNOWLEDGMENTS

I would like to thank my advisors, Dr. Donald M. Chiarulli, and Dr. Steven P. Levitan for supporting me in my work for this thesis. Without their guidance and assistance I would not have produced a document as cogent as the one before you. I would also like to thank Yan Fang for helping me on many occasions when I felt my work was stagnating.

I especially want to give thanks to my family for their patience, support and love; without them, this work means considerably less to me.

1.0 INTRODUCTION

Curiosity and the drive to solve problems has been the driving force behind science since humankind's inception. Since the dawn of computers, a lot of effort has gone into solving problems using automated computation and logic. The drive to make computers smaller, faster and more power efficient has led to an insane leap forward technologically. The technological backbone behind computers since approximately 1960 has been the use of transistors. A transistor when used in CMOS circuitry is a switch which allows current either to flow, or not flow, between a source node and a drain node depending on the state of the input. Using these as the basis for digital logic design, computers have been built for decades. Due to the high consumer demand for faster and bigger computers, there has been and continues to be tremendous funding to improve the performance of transistors. As a result, transistors are currently commercially synthesized nearly 500 times smaller (and as a result faster, since current has less distance to travel) since the first microchip was created in 1971 [1] [2] [3]. However, using transistors as digital logic building blocks represents just one of many families of computational paradigms; another family of computation using devices is called non-Boolean computation.

This work explores the use of coupled oscillators to perform non-Boolean computation. By non-Boolean computation, for the purposes of discussion in this thesis, I mean a set of computations that are not based on Boolean operators OR, AND, etc. Algorithms which can

make use of such computations include pattern recognition, pattern reconstruction, and analog processing. Applications for the using of non-Boolean computation are vast, but in this work I focus on their use in image processing, specifically the use of coupled oscillator models to accelerate currently used image processing algorithms.

1.1 MOTIVATION

As computers have trended toward a denser placement of circuit elements, many design challenges have endangered the scalability of traditional digital architectures and von Neumann computational paradigms. The main problem that has arisen in recent years has been increasing power consumption per chip area and by extension an inexorably increasing need for heat dissipation. Economically minded processor designs do not run at state-of-the-art peak clock rates because the heat generated by the processor cannot be dissipated efficiently enough to rationalize the relatively insignificant gain in performance from these efforts. Specifically, a linear increase in the clock rate of a logical circuit corresponds to a quadratic increase in the power dissipation needs [4]. Logically, comparatively compact processors end up having a slower clock rate to lessen the energy consumption which generally leads to slower calculation, while processors with faster clock speeds end up spending a large budget in solving the design challenge of heat dissipation. Substantial research has been done to solve the problem of heat dissipation, in order to allow for the gain in performance by increasing clock speed [5] [6] [7]. However, current trends suggest that the rate of increase in clock speeds is slowing, and current research trends indicate that researchers are predicting a tapering off of the rapid decrease in transistor size.

This theoretical limitation of current digital hardware technology motivates new research which has focused on novel hardware ideas to perform non-Boolean computation for existing applications. Many applications that can make use of non-Boolean computations are located within the field of Artificial Intelligence (AI), with the examples of non-Boolean computations including analog arithmetic, and pattern matching. One thing to note about the concept of non-Boolean computation is that there is not necessarily a deterministic output from a given set of inputs, meaning that two sets of identical inputs do not necessarily yield sets of identical outputs, but rather sets of close-to-identical outputs. The justification of using such computations is twofold. First, AI often deals with non-determinism inherent in many algorithms, which attempt to model the unpredictable output of the human brain. Therefore, having potential non-determinism in the calculations attempting to mimic a non-deterministic algorithm may not be disadvantageous. Secondly, many algorithms in Computer Science solve extremely computationally intensive problems, in which an exact solution is not possible given the time constraints of the application. To overcome the time constraints of such a problem, researchers frequently develop solutions which produce near-optimal solutions in a comparatively small time span, though still computationally expensive themselves. However, these near-optimal solutions are guesses themselves, and therefore are not guaranteed to be the perfect solution. In such algorithms, non-deterministic calculation may be used because exact computation in an algorithm that does not guarantee an optimal solution may not always be necessary.

Boolean computation is hitting a theoretical limit as to how rapidly and efficiently the digital hardware can perform them. In contrast, hardware devices capable of performing subsets of non-Boolean computations are hypothesized to have a much higher theoretical ceiling in terms of computational speed and power efficiency [8]. Therefore, if non-Boolean devices can achieve

close-to-ideal results on an AI algorithm with much faster speed and lower power consumption, then the tradeoff between exactness of calculation and speed/power efficiency should be considered. For instance, one such application which has been at the forefront of AI research has been that of image processing and more specifically image recognition. Real-time image recognition is currently a design challenge for modern researchers in the field, as many image processing algorithms are too complicated to perform in real time. It is hypothesized that a non-Boolean approach to solving image processing algorithms is an answer to the large power-throughput product of Boolean approaches [8]. While there have been many proposed non-Boolean devices, the one researched and discussed in this work is oscillators, and more specifically, coupled oscillators. Oscillators are defined here as any analog device that changes value periodically in time consistently and unchangingly when uninfluenced by external factors. Recently, researchers have had success in making coupled oscillators act as a pattern matching computational device [9] [10]. In this work, I propose a hardware accelerator to accomplish common image processing computations using coupled oscillators as the backbone.

1.2 PROBLEM STATEMENT

One of the classes of image processing algorithms is called an Image Processing Pipeline. For the purposes of discussion in my thesis, I define Image Processing Pipelines (IPPs) to be algorithmic constructions built to iteratively modify an input image into a series of abstractions for the purposes of decoding its contents into a higher level representation. There have been many proposed IPPs, varying in both physical construction, and in algorithmic paradigm, but by and large these designs have been based with the assumption of using hardware capable of

performing only Boolean computation [11] [12] [13] [14]. Using coupled oscillators, this work aims to utilize the benefits gained from this novel hardware design to increase performance both in terms of speed and in power in the HMAX based IPP I explore, while minimally affecting the accuracy.

My work provides empirical evidence that inserting models which represent the behavior of coupled oscillators into an existing IPP algorithm does not compromise the accuracy of the image recognition, while at the same time provides a methodology for future work of inserting oscillators into an HMAX IPP design in order to create faster, more power efficient architectures.

My hypothesis for this work is that coupled oscillator models can successfully perform computations that improve the efficiency of the HMAX architecture. The work that I do to prove this is to construct a benchmark IPP based on the HMAX design, and construct a modified version of the algorithm which uses coupled oscillator models as a primary computation. I compare the accuracy of the classification percentages between the two implementations to show the algorithmic effectiveness of the oscillator. Finally, I incentivize the oscillator-based implementation based upon research evidence that using coupled oscillators will improve performance of calculation over CMOS implementations in both speed and power. For this thesis, I do not consider the frontend part of image recognition, called saliency, and only consider the backend and how to accelerate image classification with oscillators. These terms are defined in the following chapters.

1.3 THESIS ORGANIZATION

In Chapter **2.0 Background and Related Work**, I explore previous work related to both oscillators and IPPs. I discuss the general history of IPPs, and how people have gone about solving the problems inherent in their design. Then, I discuss previous implementations of IPPs, and how they solve the problem of image extraction and classification. The next section explores the hardware accelerators which have been made in the past to accelerate IPP computation. Finally, I discuss previous work on performing non-Boolean computations using oscillators.

In Chapter **3.0 Using GRBF as an Oscillator DoM Model** I discuss models which can be used to mimic potential behaviors of oscillators. I look at both theoretical and empirical models, and discuss which model I focus on for the purpose of constructing an IPP based on oscillator models.

In Chapter **4.0 HMAX**, I explore a biologically inspired IPP. As a baseline, I construct a benchmark IPP based on a state-of-the-art implementation of an IPP, and then I modify it to use oscillator models instead of its traditional computations. I run tests to tune the parameters of the oscillators to perform the algorithm as accurately as possible, so that the classification performance is comparable with benchmark IPP.

In Chapter **5.0 HMAX Modification Discussion**, I discuss the potential speedup which may arise from these changes, and also the power savings which are expected from using coupled oscillators to perform computation.

Chapter **6.0 Conclusion, Summary, and future work** summarizes my work, I draw conclusions based upon my research, and I discuss future directions based on the results that I found.

2.0 BACKGROUND AND RELATED WORK

In this chapter, I outline previous research that has led to my work. Specifically, I discuss a brief history of image processing, and the problems that people face in the field. Next, I discuss the specific algorithms used in Image Processing Pipelines (IPPs), and explain the implementation of the IPP algorithm that I am most interested in for my work. In the next section, I discuss research that was done to accelerate IPPs beyond the speed that a typical CPU can achieve. Research has largely been in the realm of GPU and FPGA based implementations, which are Boolean-based processing implementations. This leads to the last section in this chapter, which discusses oscillators. Namely, I focus on theoretical models as well as previous research efforts to create functional circuitry to use oscillators as a computational platform.

2.1 IMAGE PROCESSING BACKGROUND

For this work, we consider image processing as the method of taking an image, or series of images as input and decoding the images into information of a higher level representation. For example, an image contains several objects, and an image processing algorithm tries to detect and determine the identity (class) of each object. As a concrete example, Figure 1 is from the Stanford Tower Image Database and shows bounding boxes around objects of interest [15]. In general, the IPPs discussed in this work are composed of two fundamental stages of processing:



Figure 1 – Sample image from Stanford Tower Datasets with bounding boxes around objects of interest

frontend processing and backend processing [16]. The frontend of image processing is the stage in which the bounding boxes around objects are drawn and these sections of the image, which we call *image chips*, are extracted (the “where”), and the backend is where the type of object inside of the bounding box is determined (the “what”) [16].

In the frontend, in order to determine where interesting objects are, a saliency analysis is typically done on the image. The saliency of a region in the image is a loosely defined term which means “how interesting is this part of the image?”. The goal of saliency analysis is to have it output high saliency for parts of the image that contain objects of interest, and to have a low saliency for as much of the rest of the image as possible. This way, only the parts of the image with high saliency are the objects of interest and are passed to the backend for determining what is the object of interest. There are many ways of performing saliency analysis, two of which are discussed in the next section. The backend is responsible for determining the type of object contained in the image chips that are output from the front end. There are many

algorithms for performing the backend computations, some of which are discussed in the next section.

2.2 PREVIOUS WORK ON IMAGE PROCESSING

In this section, I discuss prior research done for image processing including the frontend and backend. The frontend discussion provides context to my work on the backend and gives a part of a more complete picture of the construction of an overall IPP. For the backend discussion, which deals with classifying objects from an image, I talk about testing and training the algorithm separately, as they are two distinct algorithms. Semantically, I often call the testing algorithm the “feed forward” algorithm, since conceptually we push data through the structure, and the training methods are done in a feed forward or backpropagation style, depending on the algorithm. These terms are defined more clearly in the section in which they are discussed.

2.2.1 Frontend Previous Work

This section explores the work done on the frontend part of the IPP. The frontend is responsible for taking a large image, and decomposing the image into segments, called objects of interest, which, hopefully, are parts of the image that we want to classify. The idea behind this step is many-fold, however the main reason that this step is important is that it decreases the amount of work that needs to be performed by the backend of the IPP. Saliency-based extraction on the front end of the IPP creates modularity between determining the “where” and the “what” of object recognition. As I will discuss, this has the benefit of increasing IPP throughput, but also it

is a biologically inspired model as the visual cortex in the brain has two separate functional units to determine the “what” (ventral stream) and the “where” (dorsal stream) [17] [18]. By extracting small segments of the image, called image chips, the saliency detector eliminates the need for the IPP feature extractor and classifier (backend) to analyze large sections of input images, which has been shown to speedup IPP throughput [19] [20] [21] [22] [23]. As an example of saliency detection, Figure 1 shows the bounding boxes drawn around objects of interest. The alternative to this methodology is for every possible section of the input image to be classified by the backend, which 1) decreases throughput and 2) puts more stress on the classifier to eliminate false positives on object classification. There have been many proposed approaches to performing saliency analysis. In this section I discuss a model based on center-surround differences with normalization, and a model based on spatial Fast Fourier Transforms (FFTs) to extract conspicuous regions.

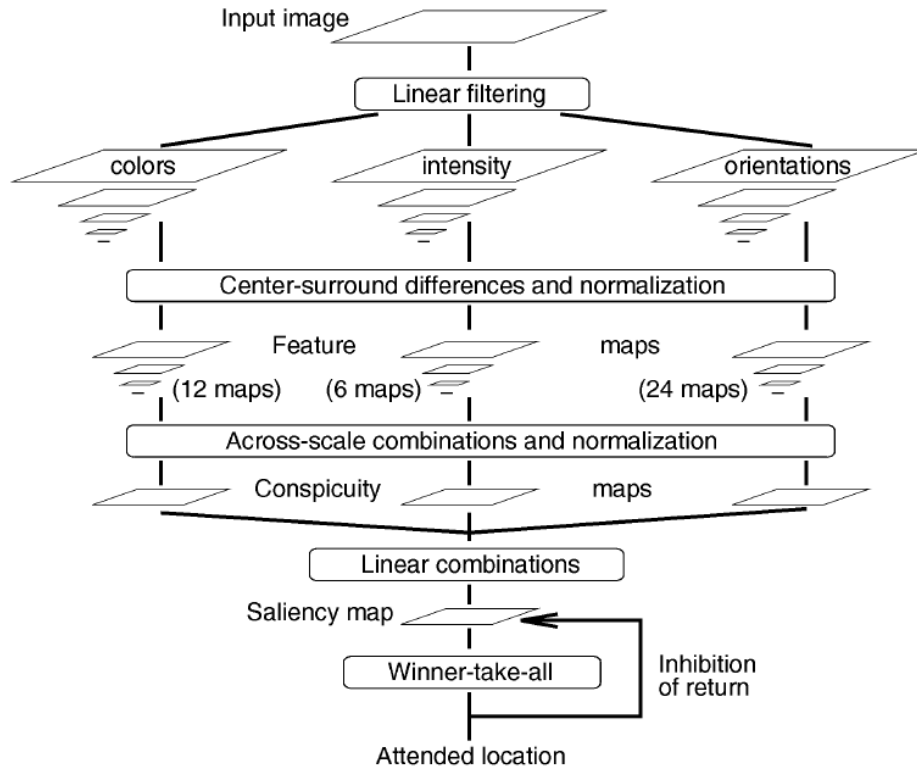


Figure 2 - © 1998 IEEE - general architecture of Itti's saliency model (Itti, Kock, & Niebur, 1998)

Some of the early work on saliency, and its use for scene analysis was done by Laurent Itti. This early work can be summarized by his research in [20]. The work is inspired by neuroscience in the fact that Itti remarks how studies had shown how primate's brains 1) are the most powerful image processing systems known, and 2) have separate dorsal and ventral streams for determining the location of objects and type of objects, respectively. This inspired his work to have a system which mimics a neurological phenomenon of using only local receptors to achieve more global coordination. Using this biologically inspired approach, he built the following saliency model, which is shown in Figure 2.

Itti creates 9 image scales, and uses linear filtering to separate the image into 3 channels: a color channel, an intensity channel, and an orientation channel. This step is shown in the linear filtering stage in Figure 2. The next step is to use the center-surround differences and

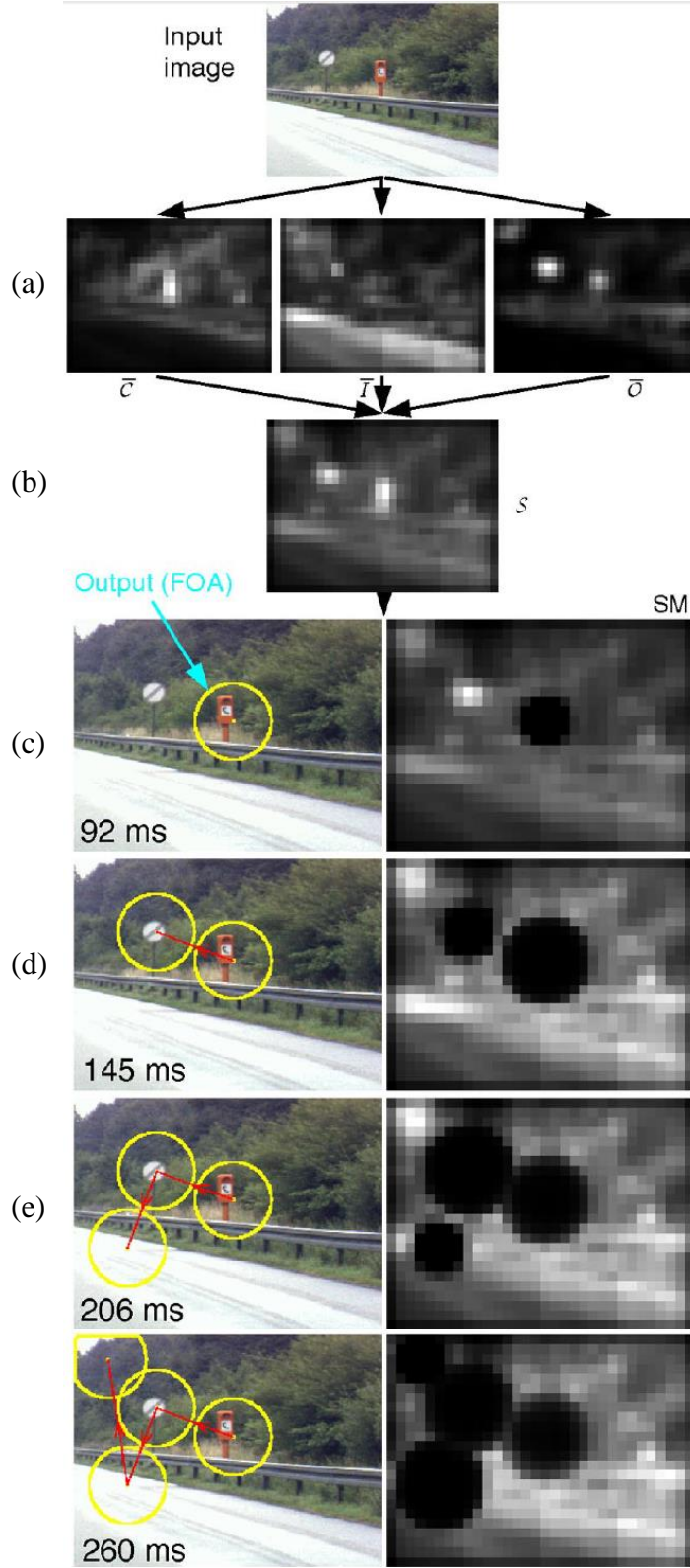


Figure 3 - ©1998 IEEE– Example of the operation with natural image. First, three saliency channels are summed to create total saliency, and areas are chosen, and then inhibited in cyclical fashion to determine highly saliency regions of interest [20].

normalization computation, which is calculated based upon a point by point difference between the pixel values in their respective channels at a smaller scale, and surrounding pixels values at a larger scale, shown in the “center-surround differences and normalization” part of Figure 2. A larger sum of differences calculated in this way corresponds to a larger the saliency at this location. Using multiple pairs of scales to calculate the center-surround differences improves performance. This computation is performed for all filters, and then pooled from scales down to a flat saliency model, shown in the two steps across scale and normalization, and linear combinations in Figure 2. The last step is a summation of the saliencies of the filters, and a winner-take-all (WTA)-meaning “largest saliency wins”-approach to which chip is looked at first.

Then, surrounding saliencies in the

saliency map are reduced for this image chip chosen and the process is repeated some number of times. All of the computations discussed are local, until the final WTA calculation. The WTA is modeled as a firing mechanism in a local neuron, which is then inhibited from firing again, which has also been shown to occur in the primate visual cortex. This cyclical WTA process is shown in Figure 2 at the bottom of the figure with the “winner-take-all” box and “Inhibition of return” arrow.

As an example, consider Figure 3. The original input image is filtered into a color, intensity, and orientation channels, and Itti creates 9 image scales of each of the 3 channels. The saliency of each small scale pixel of the channels is computed based upon sum of the differences between the pixel at the small scale and the pixels surrounding the corresponding location at larger scales of the image. The number of scales to consider and the size of the neighborhood of the center-surround computation are parameters of the algorithm. The saliency maps of each of the channels are shown in (a) in Figure 3. Step (b) represents the next step of the algorithm, which sums the saliencies of the 3 channels, condenses the 9 scales into one scale, and normalizes the results to create one complete saliency map of the image. The WTA method then chooses the region with the largest saliency as an image chip, which is the yellow circle depicted in (c). Itti then inhibits the saliency map at the surrounding location of the image chip chosen, re-normalizes the saliency map, then repeats the process described for (c) some set number of times. Steps (d), (e), and (f) are continuations of this process, which output 3 more areas of interest within the original input image.

Itti’s model solves a lot of problems, in that it gives the backend of an image processing algorithm specific locations of interest. However, the number of locations in the image to analyze is data dependent and thus does not provide a data-independent way of determining

which scenes have greater or fewer number objects of interest. To solve this problem, work has emerged since, which looks at taking spatial frequency analyses to determine the saliency of regions.

This way of determining salient regions is done by performing a FFT analysis on the spatial input of an image channel. The work done by Hou and Zang in [19] is a starting point for this work in saliency, which came about 9 years after the work done by Itti. This work uses an observation that analyzing the response swept over the spatial frequency of the input image has a characterizable shape for all natural images. The saliency of a given region is then defined as the local difference between this curve and the characterized curve for natural images. The method by the researchers was to average the FFT over many natural images, then using this for a baseline, run test images to discover regions of interest in the image, and use these to extract image chips.

The extraction method is similar to the method used by Itti in [20], however, there are more defined regions of interest, and because there is no cyclical behavior and inhibiting saliency for given regions there can be different numbers of objects of interest in a given image. However, this approach is not as biologically driven, and is a more recent development, so less research has been done to expand upon this path.

The work done for saliency detection and image chip extraction in this research is interesting, and worth exploring, but is beyond the scope of the work that I do in this thesis. In my work, I focus on the backend of the IPP. I assume that the input to my algorithm is an extracted image chip, with which I have to determine the class of the image chip. The rest of this chapter discusses previous work done in this area and acts as an introduction to the work for my thesis.

2.2.2 HMAX

My work is based upon a backend IPP called HMAX. HMAX, in combination with a classifier at the end of the pipeline, composes the backend of an IPP created by Mutch and Lowe [24].

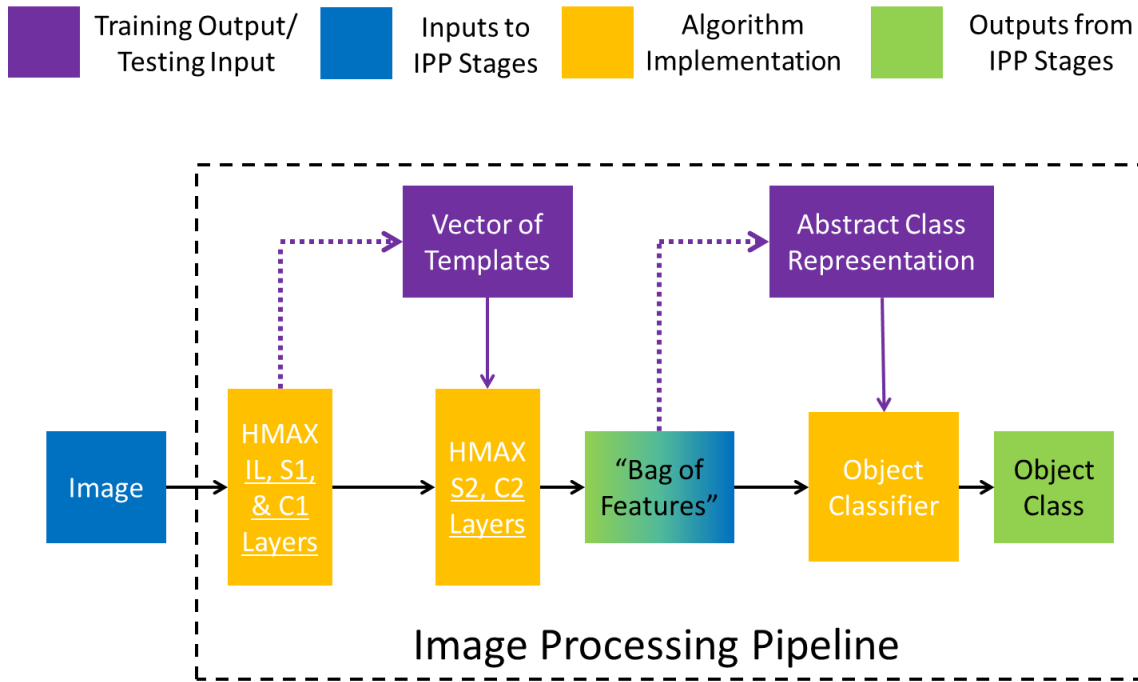


Figure 4 – Data flow of the HMAX-based IPP

The HMAX IPP was developed in 2006 by Mutch and Lowe, who used the biologically feasible HMAX feature extractor with a Support Vector Machine (SVM) backend to perform image classification [24]. The input to their IPP is an image chip which is a member of a class to be determined. Figure 4 shows the high level overview of the data flow of the IPP. HMAX is broken into 5 distinct layers, Image Layer (IL), S1 Layer, C1 Layer, S2 Layer, and C2 Layer. The IL does scale invariance, the S1 layer does edge detection, the C1 layer performs pooling and max filtering, the S2 layer does template matching, and the C2 layer performs a max operation. Figure 5 shows these 5 layers in more detail.

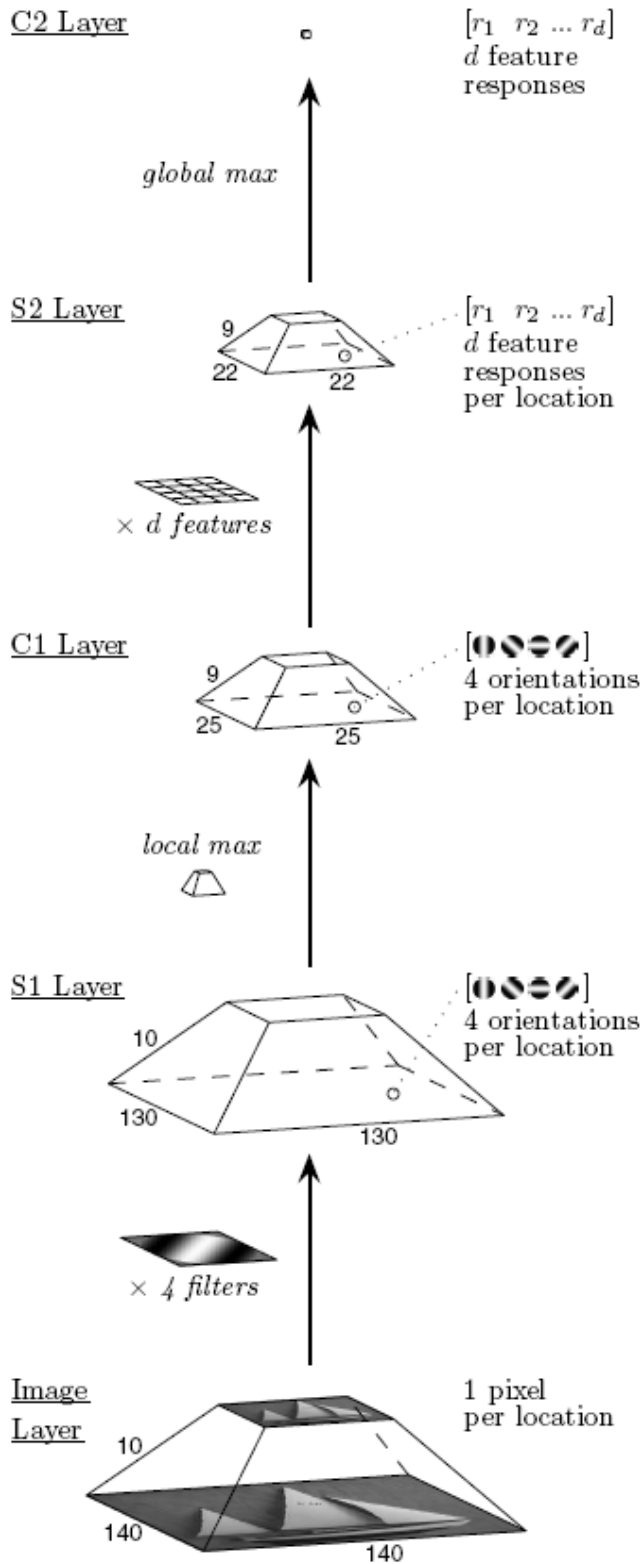


Figure 5 – ©2006 IEEE - Pictorial representation of HMAX

[24]

The IL is responsible for scale invariance, which is the concept that no matter how large an object is it should be recognized by the IPP. To achieve scale invariance, the IL rescales the input images to multiple scales, and passes the output to the rest of the IPP. The number of scales of the images, the largest size of an image, and how much smaller each subsequent scale are, are all parameterizable. Mutch and Lowe's implementation has 10 scales with the largest scale being 140x140, and each subsequent scale decreasing by a factor of $2^{1/4}$.

Moving up in Figure 5, the S1 Layer is responsible for the preprocessing of the image, where, in this case, preprocessing means edge detection. Edge detection is the process in which a filter designed to accentuate edges is convolved over an input image.

For HMAX, this specific filter is called a

Gabor Filter [25] [26]. In order to detect various orientations of edges in the different scales of the image, 4 angles of Gabor Filters are used each of size 11×11 . These 4 outputs of the edge detection convolutions are saved and passed to the next level.

Moving upwards again, the C1 Layer performs max/pooling operations on the output of the previous layer. This means that between two adjacent scales of the images output from the S1 Layer, the C1 Layer passes a max filter over a $10 \times 10 \times 2$ region of the pyramid, and downsamples by a factor of 5. The idea of downsampling is a common theme among IPPs, as it is biologically inspired from neuron behavior in the brain [27].

Next, the S2 Layer performs template matching between known features extracted during the training phase, which is discussed later, and each location in the output pyramid from the C1 Layer. The result is d pyramids of Degrees of Match (DoMs), where each DoM represents the location result of the d template matches on each C1 pyramid output. The idea is to look for the features in any location in the image, to achieve spatial invariance. Spatial invariance is the idea that the location of the object within an image should not matter for detection or classification. In other words, an object that is located in the center of the image should be as easily classified as an object that is located on the edge of the image. The DoM pyramids are output from this layer and input to the final layer of HMAX.

Similar to the C1 layer, the C2 Layer performs a maximum operation on all of the DoMs in each of the d pyramids output from the previous layer. The output of this layer is a d dimensional vector of values, each representing the maximum degree of match of a feature at any location of the preprocessed image pyramid.

The last step in Mutch and Lowe's IPP is the use of a Support Vector Machine (SVM), which is described in section **2.2.4 Feature Vector Classifiers**.

2.2.3 Convolutional Neural Networks

A second type of backend in IPPs can be generally called Convolutional Neural Networks. Convolutional Neural Networks (CNNs) are a byproduct of years of research and a solution to existing limitations of earlier neural network designs. There have been many neural network designs developed since their inception, such as linear and non-linear neural networks (NNs) and CNNs, which each specialize in a particular area of neural computing. This section introduces the ideas behind the inception of neural computing and why researchers continue to research them today.

In 1962, Hubel and Wiesel discovered locally-sensitive, orientation-selective neurons in a cat's visual cortex, which was a hint that larger intelligence can be achieved through smaller, narrow sighted individual pieces. Around the same time, Widrow and Hoff developed the first successful adaptive neural network with local receptive fields. Essentially, this started an entire research field in computer science based on the work by Widrow and Hoff inspired by the biologically inspired models found by Hubel and Wiesel [28] [29].

Over the decades, neural networks have been used as object classifiers, fuzzy logic control and decision systems, function mappers, associative memories, image classifiers, face detectors, and many more [30] [31] [32] [11] [33] [34] [35]. NNs, while they are nearly universal in their uses, are not specialized to one particular function.

On the other hand, CNNs are a specialized version of NNs that are useful for image processing as they inherently include spatial locality in their architecture [30] [14]. Further, training that uses CNN backpropagation is robust, and can tolerate a wide range of learning rates and filter sizes [11] [36]. CNNs are also useful when preprocessing of an image chip is not desired before classification, which is used by many NN implementations of image recognition

[11] [37]. CNN backpropagation essentially creates its own feature extractor, thus features are not chosen beforehand like in HMAX [14]. Like its predecessors, CNNs are generally based on the backpropagation training methodology, which is discussed in the subsequent subsections.

Here I discuss an implementation of a CNN, which uses several fully connected hidden layers of convolutions. Typically in CNNs, there is one input matrix, with some number of neurons per layer, and some number of layers. At each “C” layer, Equation 1 below determines the matrix outputs for a given layer, given the set of inputs at a given layer. out_j is the j^{th} matrix of the output of the layer, in_i is the i^{th} matrix of the input of the layer, and k_{ij} is the filter for the connection from in_i to out_j . The symbol \bullet represents a convolution, and $\text{sigm}()$ represents a sigmoid function. This can be implemented as a number of different functions, but an example of a sigmoid function is shown in Equation 2, with a plot of this function in Figure 6.

$$out_j = \text{sigm}\left(\sum_i k_{ij} \bullet in_i\right) \quad \text{Equation 1}$$

$$\text{sigm}(x) = \frac{1}{1 + e^{-x}} \quad \text{Equation 2}$$

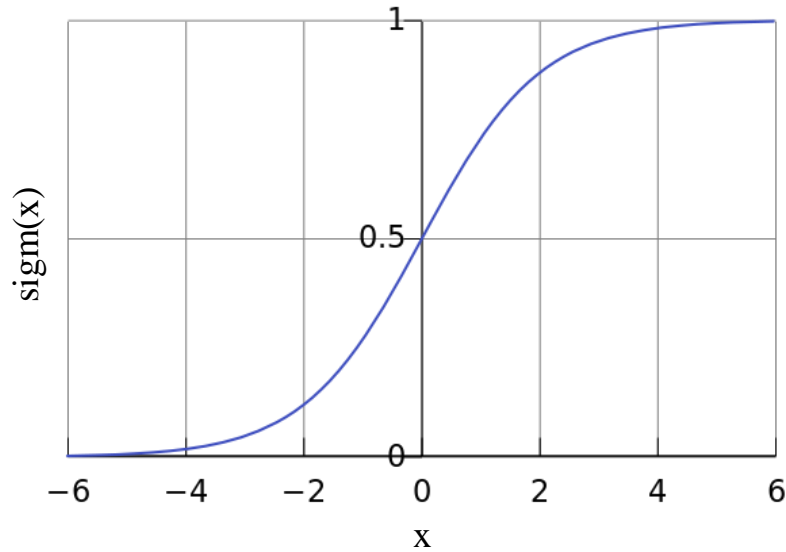


Figure 6 – Sigmoid function output shape

Another type of layer often employed in a CNN is an “S” layer. S layers are usually alternated with C layers. An S layer pools data by averaging pixels within a certain region by filling the kernel of the layer with all ‘1’s and convolving it with the input image. The result is a “blurring” of the input image. The output of the S layer is a subsampled version of this blurred version of the input image. Typically the image is subsampled by a factor of between 2 and 6, depending on the size of the filters. Note that this layer is not trained and the filter always consists of all ‘1’s, since it should always act as an averaging filter.

2.2.4 Feature Vector Classifiers

In this section, I discuss three methods used to classify images by their feature vectors: Support Vector Machines, Neural Networks, and nearest neighbor classification.

Support Vector Machines are classifiers that take as input a weight vector of known length, and output a class [38] [39]. An SVM classifier consists of a training phase and a testing phase. In the training phase, a standard SVM classifier takes a set of vectors, which are

predesignated into 2 classes. Also in the training phase, a hyperplane is drawn between the points in the two classes. In the testing phase, an input vector is classified based upon which side of the hyperplane the point falls. For the IPP, the vectors used for classification are the outputs from HMAX. In order to facilitate multiple classes (more than 2 classes), we must create many SVMs. There are many methods, but the method used by Mutch and Lowe [24] is the same method that I implement here, called one-versus-one multiclass SVM. The idea is to build a SVM hyperplane between each pair of classes in the test data, then perform a max-wins voting strategy in which every individual SVM classifies the input into one of two classes. The class which has the most “votes” after all SVMs have classified the test vector is the class which we chose as the label for the image.

Nearest neighbor classification is where the classifier remembers the feature vectors from all of the training images that it has seen in N dimensional space. The testing phase of classification, then, simply performs a distance computation between the test image’s feature vector, and all of the training images’ feature vectors. The training feature vector which is closest to the testing feature vector is determined to be the winner of the classification, and the testing image is classified as the same class as the corresponding training image. Figure 7 shows a pictorial representation of a training set, and the difference between how an SVM classifier classifies test images versus how a nearest neighbor classifier classifies test images, using 2D training vectors.

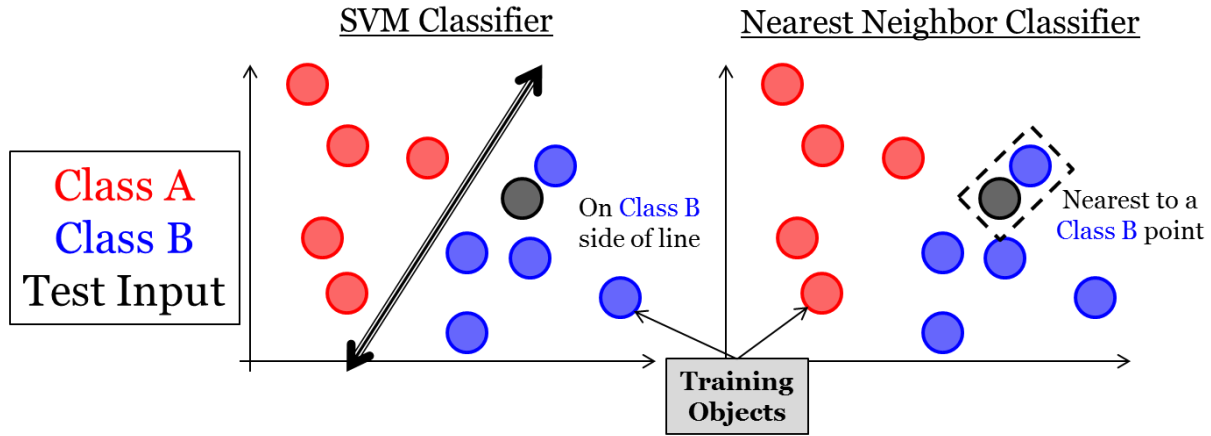


Figure 7 – SVM classifier versus nearest neighbor classifier: classifies the test input as class B in 2D space

Neural Networks, or NNs, while used for many things, are often used as classifiers [14] [33]. The input is the same as for an SVM, namely a vector of weights, but the output is different. In an SVM, multiclass classification comes from creating multiple SVM classifiers, but NN classification directly implements multiclass classification. The training phase for NNs is discussed in section **2.2.5 Training Methodologies**, but here I discuss the testing methodology for NNs as classifiers. The specific NN on which I focus in this section has complete connectivity between the input vector and output vector, which is common in classification. The input vector is the feature vector which is output from the feature extractor section of the IPP. In this chapter, I discuss two different forms of feature extractors, HMAX and CNNs. The output from both can be thought of as feature vectors. The output of the CNN is an N dimensional vector which represents the confidences for each class to which the image might belong. The connections from the input to the output are the weights that are associated with each of the features, with each weight representing how heavily this feature indicates which class it could be. Equation 3 determines the output for each of the N output elements from the F input elements.

out_j is the j^{th} element of the output vector, in_i is the i^{th} element of the input vector, and w_{ij} is the weight of the connection from in_i to out_j .

$$out_j = \sum_{i=1}^F w_{ij} \times in_i \quad \text{Equation 3}$$

In general, the connectivity of the NN can be sparser, there can be more layers of computation between the input and output, and the transfer function can be non-linear. In this case, it is a fully connected NN with one layer, and the transfer function is a multiplication. Figure 8 is a picture of such a network where F is 5, and N is 3. Each line in the NN graph represents a multiplication, with the green output nodes performing a summation of the incoming edges.

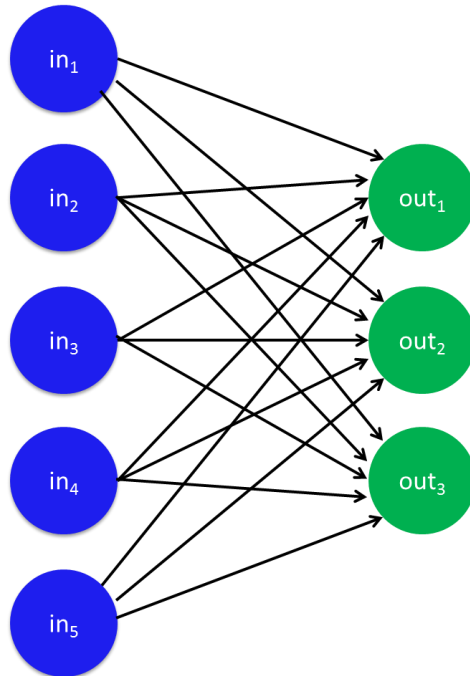


Figure 8 – Neural Network classifier with 5 features and 3 output classes

2.2.5 Training Methodologies

In this section, I discuss the difference in methodology between backpropagation and feed forward training. Backpropagation is the training paradigm used largely in NNs and CNNs, where the feed forward training is the method which is used in HMAX.

Backpropagation is the process of setting the state of the IPP by determining how to change the weights, filters, or templates in order to correct the error that was found from a run of the current state of the IPP. This is done differently depending on the type of network we are training, and there are many ways of performing backpropagation for each network. In this section, I discuss one method of how to perform backpropagation training on the NN I discussed, as well as how to use backpropagation to train the CNN I presented earlier.

For a given layer, the input to the backpropagation for this step is the Δout desired, and the current state of the filters and input (which was calculated during the feed forward part of the process from the previous layer(s)). Equation 4 gives the formula for the computation of Δk_{ij} and Δin_j based on the Δout_i desired and the current state of the IPP (in_j and k_{ij}).

$$\begin{aligned}\Delta in_j &= \sum_i \Delta out_i \bullet_f rot180(k_{ij}) \\ \Delta k_{ij} &= rot180(in_j) \bullet_v \Delta out_i / fanout\end{aligned}\tag{Equation 4}$$

Δk is used to determine the new state of the CNN and Δin is used to continue backpropagation to the previous layer. Like in previous equations, the \bullet denotes a convolution, however in this case a \bullet_f denotes a full convolution, and a \bullet_v denotes regular convolution (over the valid range). To compute \bullet_f , a buffer of size $n-1$ is put on each side of the original input matrix, then a \bullet_v convolution is performed on this modified input matrix with the original filter

(of size $n \times n$). The value in the buffered region around the modified input matrix is extrapolated from the nearest border on the original input matrix. In other words, the value of a pixel in the border is identical to the closest associated pixel in the original input matrix. The modified input image is of size $2(n-1) \times 2(n-1)$ larger than the original input image, which is depicted in FIG.

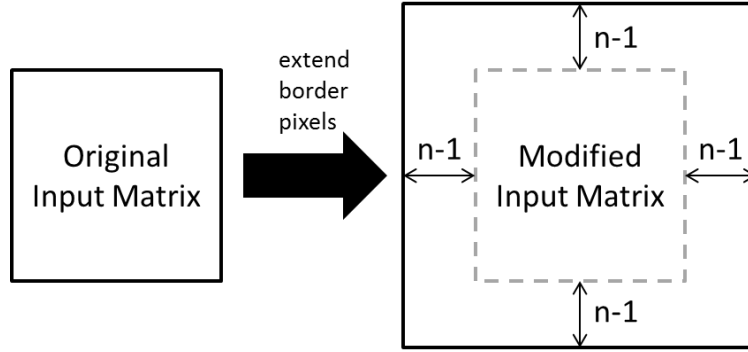


Figure 9 – Example of how to modify input image to perform a full convolution operation (\bullet_f)

Note that a Δin_j for one layer is a Δout_i of the previous layer. Each layer computes the Δin and Δk for each of the states of the CNN, then simultaneously updates each of the filters (the input to a given layer is determined based upon the previous layers' filters). The filters are updated based upon the following formula.

$$k_{ij} = k_{ij} + \alpha \times \Delta k_{ij} \quad \text{Equation 5}$$

α is a number between 0 and 1, and represents the learning rate of the system. This entire process is repeated for each training image, and is called an epoch. The process can then be iterated over for multiple times, which would constitute training the CNN for multiple epochs.

In HMAX, the training methodology is a feed forward methodology, which is split into two phases. During the earlier discussion of the feed forward process, there were two pieces missing: 1) what templates do we use in the S2 layer, and 2) which SVM hyperplane do we use between the classes for classification. Each of these is done separately.

First, we determine the templates. This is done by running all of the training images through the IL, S1 and C1 layers, then extracting a certain number of random templates from each training image. For more or less extensive training, the number of extracted templates is parameterizable. Second, using the templates we extract from the first training stage, we run the N training images through all 5 layers of HMAX and output N feature vectors to the SVM. Using the known labels and N feature vectors, we train the SVM by creating a hyperplane between each group of object classes and use these SVMs as the basis for the max-wins voting strategy discussed earlier.

2.3 PREVIOUS WORK ON IPP HARDWARE ACCELERATORS

In the past, many researchers have explored the possibility of building hardware accelerators for IPPs [40] [13] [41] [42]. In general, these have been based on Field Programmable Gate Arrays (FPGAs) and Graphical Processing Units (GPUs), which perform massively parallel Boolean computations. In each of the designs, the idea is to save time, not power with these accelerators. For performing convolutions, template matching, or calculating the output of a NN, the operations in an IPP are extremely parallelizable by nature, meaning that at each layer the computations do not rely on the output of the other. Researchers, therefore, have created programs to run on extremely parallel machines to speed up the computation.

In [40], Clement et al. developed an accelerator for CNNs by constructing a streaming architecture which computes the convolutions in pipeline-fashion. They built the program such that they could synthesize their design both in Application-Specific Integrated Circuit (ASIC) and FPGA fabrications. The ASIC implementation of their design achieved a speedup of more than 100x their benchmark CPU implementation. In [13], Al Maashri et. al. created an FPGA based accelerator on both a single FPGA platform and multi-FPGA platform to achieve speedups of up to 89X. In [42], Sabarad et al. developed an accelerator for the HMAX algorithm discussed in an earlier section. Their implementations were on both GPU and FPGA, and realized an improvement in performance-per-Watt of 14x and 33x respectively with these two implementations.

From past work, it is clear that speedup is indeed possible and that parallelization is useful for speeding up computation. However, due to the size of the accelerators and the massive number of computations per second that they are accomplishing, they also consume tremendous amounts of power. For this reason, my thesis explores the algorithmic domain of using a lower power accelerator for the computations, while still hoping to achieve similar speedup through parallelization and streamlined pipeline architectures.

2.4 NON-BOOLEAN COMPUTATION WITH OSCILLATORS

This section explores some of the theory behind coupled oscillators, the models that describe oscillator behavior and previous research that have use coupled oscillators to perform computation.

2.4.1 Empirical DoM Model

Probably, the first known discovery of the concept of synchronization of interacting systems was by Christian Huygens in 1673 [43]. He developed pendulum clocks, and realized that when sitting on the same base, the clocks would synchronize to the same phase, or exactly 180 degrees out of phase. Since then, many other models of interacting oscillatory systems have been created, among them electronic oscillators [44] [45] [46] [47] [48].

There has been a lot of research in modeling the energy transfer between coupled oscillating bodies, many of them applied to electronic circuitry. Recently, researchers have developed simple circuitry to determine a Degree of Match (DoM) function using coupled oscillators [9] [10]. By DoM, I mean the oscillators' ability to synchronize. A higher DoM indicates a better synchronization (closer to the same phase and frequency), and a lower DoM indicates a worse synchronization. The organization of the system is to have N oscillators coupled, where each oscillator's uncoupled frequency is voltage controlled, meaning that an increase or decrease in input voltage increases or decreases the oscillators' frequency, respectively. The type of oscillator explored in [9] was a CMOS ring oscillator and the DoM circuit was an integrator of the rectified coupled voltage. The researchers discovered that when the oscillators were closer to oscillating at the same frequency, the integrator would yield a higher value than when they were not at the same frequency. A sample of the output of the oscillator after a fixed time period of the integrator yields a voltage which shows monotonically decreasing responses as the input frequencies get farther away from each other. To more precisely model this behavior, the following formula was developed to describe this behavior.

$$DoM(\vec{V}) = \left[\frac{2}{N(N+1)} \left(\sum_i^{N-1} \sum_j^{i-1} \exp\left(-\frac{|v_i - v_j|^2}{\gamma}\right) + \alpha \sum \exp\left(-\frac{|v_{match} - v_i|^2}{\gamma}\right) \right) \right]^{\frac{1}{2}}$$

Equation 6 – DoM formula found empirically from a CMOS ring oscillator using an integrator on the summing node from [9]. The input to this function is a difference between the input vectors:

$$\vec{V} = \vec{X} - \vec{T}, \text{ where } \vec{X} \text{ and } \vec{T} \text{ are the input vectors to the circuit.}$$

V_{match} is a voltage that represents a difference between the vectors of 0, to avoid aliasing. Aliasing is the term used for the following phenomenon: when the difference between the individual vector elements in \vec{X} and \vec{T} are constant, then there will be a perfect match even if \vec{X} and \vec{T} are not matches. V_{match} solves this problem by inputting a voltage that represents a difference of 0 between the input vectors, to lessen the effect of aliasing. α is the variable used to determine the strength of the coupling between V_{match} and the other input voltages, which represent the difference between \vec{X} and \vec{T} . Lastly, γ serves the same purpose as γ serves in the GRBF function: to affect the spread of the bell curve given by that term.

For my work here, I use this model, or models which are similar to it, in order to emulate the behavior of the DoM circuit created in [9] and [10], so that I can develop an algorithm based upon the behavior of coupled oscillators.

3.0 USING GRBF AS AN OSCILLATOR DOM MODEL

In this section I talk about different models representing the behavior of oscillators, and models to represent the degree of match (DoM) of the oscillators, and also empirical evidence to support using a Gaussian radial basis function as a model for oscillator behaviors [9] [10].

A radial basis function (RBF) is any function that takes as input two vectors, and based upon their distance from one another, outputs a value. The distance that is used as input to the RBF can be a number of models, including Euclidean distance or Manhattan distance. In other words, the exact location of either of the vectors is not used as the computation for the output of the RBF, but only the distance between the two. Sometimes, an RBF can take as input one vector and the other point is assumed to be the origin. Another possibility is to simply pass a distance to the RBF and have it return the value. Due to its definition, there are many possible RBFs, but the one that I focus on for this work is a Gaussian RBF. A Gaussian RBF (GRBF) is a function which scales a distance, d , by a Gaussian distribution as follows.

$$GRBF(d) = e^{-\left(\frac{d^2}{\gamma}\right)} \quad \text{Equation 7}$$

The output of this function is often called a response as a result of the distance between the input vectors. γ determines the spread of the curve, where a larger γ creates a distribution

with a larger standard deviation. Namely, the standard deviation, σ , is equal to $\sqrt{\gamma/2}$. Figure 10 shows a GRBF where $\gamma=1$.

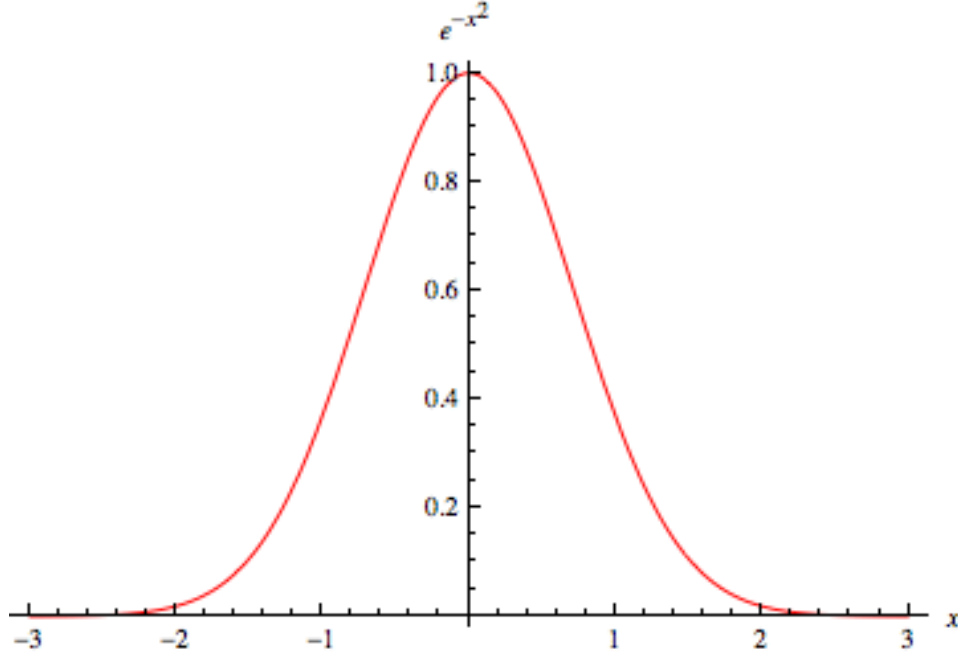


Figure 10 – GRBF with a $\gamma=1$

From section **2.4.1 Empirical DoM Model**, Equation 5 shows an empirical model for representing the degree of match between a set of oscillators. To validate the idea of using GRBF as an estimate for this model, I ran the following experiment to determine the similarity between the two.

For this experiment, I set $\alpha=0$ from equation 5 and determined if there is a roughly linear relationship between GRBF and DoM with respect to the distance between the input vectors. For the purposes of this test, I assumed that the γ for the GRBF function from Equation 7 scales as the square of the size of the input vector with respect to the γ for the DoM model from Equation 6. Figure 11 shows the linear relationship between these two models for input vectors of size 16, 32 and 64.

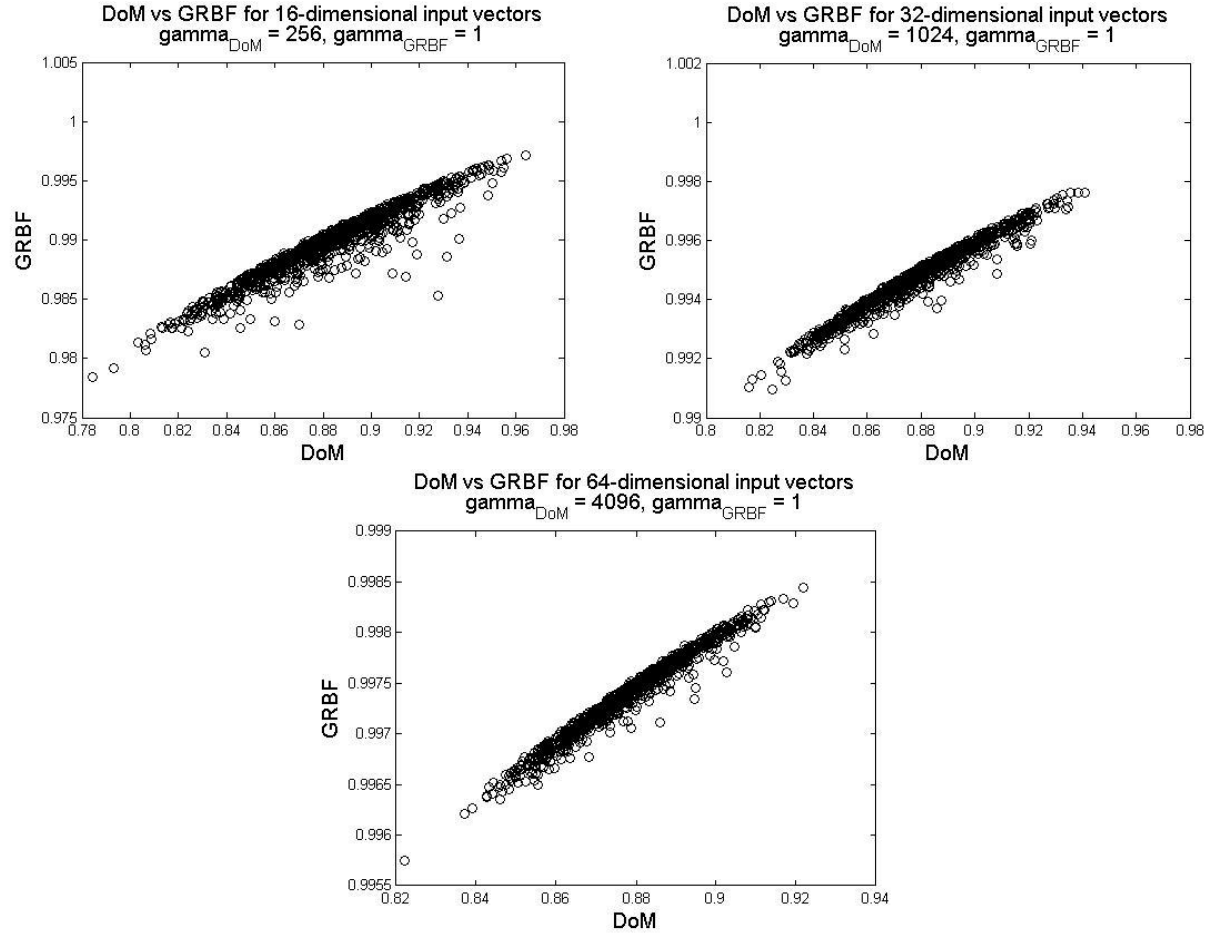


Figure 11 – Scatter plot demonstrating the linear relationship between DoM and GRBF for input vectors of size 16, 32, and 64

As a result of this experiment, it is clear that GRBF is a good model to represent the coupled oscillator behavior found in [9]. Since the actual response of other possible oscillator technologies is not well known, we will use both Equation 6 (DoM) and Equation 7 (GRBF) as two possible oscillator models when we develop our algorithms for our IPP in order to gain the potential speedup and power savings that can be offered from the coupling oscillator emerging device technologies.

4.0 HMAX

In this Chapter, my goal is to show that oscillator models used as a replacement for traditional calculations performed in IPPs is a valid approach. This chapter explores the parameters of HMAX, and discusses how we can adjust them to allow HMAX to perform image classification using oscillators as well as the traditional model. In the next chapter, I discuss the performance impact of these changes.

4.1.1 Data Set

For this work, I use the Stanford Tower Dataset [15] with the objects of interest pre-extracted. From an “assumed” front end, the following table shows examples of the 5 classes used.






Person	Cyclist	Car	Bus	Truck
				

Table 1 – Example images from Stanford Tower Dataset used as input to HMAX IPP

For the purposes of separating the dataset into testing and training sets, I take 21 images from each class and put them in the training set, for a total of 105 images for the training set. For the test set, I take 20 different images per class and put them into a test set. When I give classification percentages, I am referring to the percent of the testing images that were classified correctly after the IPP is trained using the training images.

4.2 BASELINE IMPLEMENTATION

Using the parameters from Mutch and Lowe's work from 2006, I was able to construct a representation of their IPP using Matlab as my platform. There are many parameters in the layers of HMAX, but using theirs as the default parameters, I was able to achieve a classification percentage of up to 97%, and as low as 93%, depending on the templates that are randomly extracted during the training phase. In this chapter, I discuss the various parameters of interest in the IPP, as well as a few modifications to Mutch and Lowe's IPP to construct a benchmark IPP based on the traditional HMAX implementation. The goal of this section is to maximize the performance of the benchmark HMAX IPP by adjusting the parameters at each layer in HMAX, generate benchmark results, and then create an oscillator based IPP to attempt to reproduce the results achieved from the benchmark IPP.

4.2.1 HMAX Parameters

For the Image Layer, the parameters are the number of image scales that are chosen and also how far apart the scales are. For this, I wanted to minimize the number of image layers necessary,

since for this dataset, scale invariance should not be important due to the common scale of all of the input images. To show this, I ran a test of 10 runs of the IPP with each of 2 through 10 image layer scales used, and recorded the classification percentage. Figure 12 shows the relationship between number of images layers and the classification percentage. The error bars on the classification percentage represent a standard deviation of the classification percentages over 10 runs.

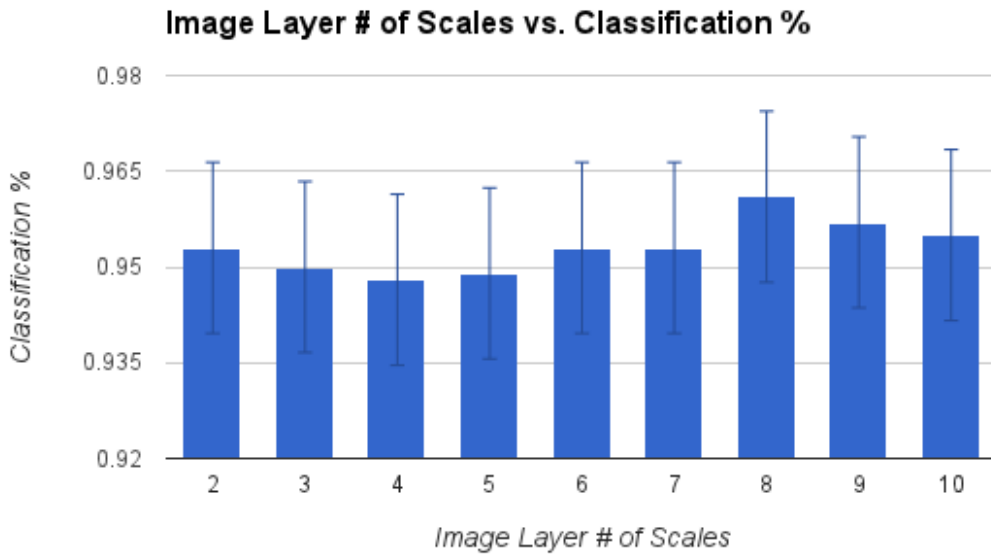


Figure 12 - Number of Scales output by the Image Layer versus Classification Percentage

This result shows that the classification percentage is not directly affected by the number of image scales produced by the image layer of the HMAX IPP. Therefore, in order to reduce the amount of computation necessary by my IPP, I reduce the number of scales produced by the image layer to 2. Note that due to the construction of the C1 layer, 2 layers are necessary in order for the biologically inspired portion of max filtering and pooling to correctly occur.

The other parameter is the scale factor between the image scales output from the IL. Mutch and Lowe decided to use a scale factor of $(1 / 2^{1/4})$, which I also use in this work. Due to

the lack of scale invariance needed, I kept this factor the same as in their work since it should not have any drastic impact on the classification performance of the IPP.

In the S1 layer, there are several parameters to adjust: the type of filter used (Gabor Filter), the size of the filter (11x11), as well as how many filter orientations to use (4). The type of filter determines the pixel values of the filter, which is chosen statically before the IPP is trained. Second, the size of the filter determines how many dot products are necessary for the convolution in the S1 layer, as well as the size of each dot product. Increasing the filter size increases the size of the dot product, but also decreases the number of dot products necessary to perform. The number of filter orientations is linearly proportional to the amount of work necessary. Following Mutch and Lowe, I kept this parameter constant at 4 orientations for rotational invariance.

The Gabor Filter that I use in this work uses the same filter as from [24], where $\lambda=5.6$, and $\sigma=4.5$. To demonstrate the output of a Gabor Filtering process using 4 rotational channels each of size $N \times N$, I construct the following table to show how the filter acts as an edge detection algorithm with respect to each of the 5 classes. These images are the same images as from Table 1. Remember that the output from the S1 layer is each of the 4 filters passed separately, not the summation of the filters. The summation output in the last row of the table is just a visual illustration of the edge detection ability of the filter.






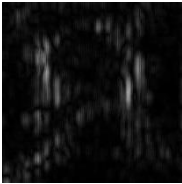
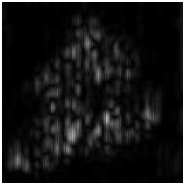
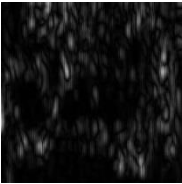

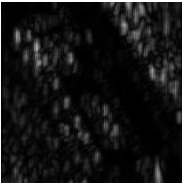
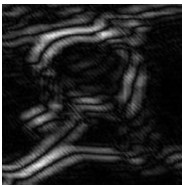
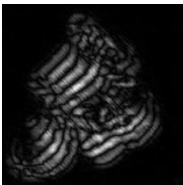



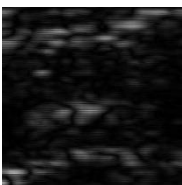
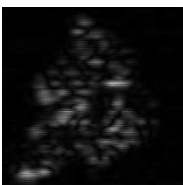
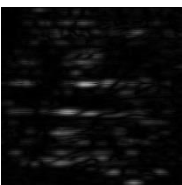
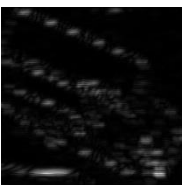
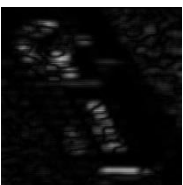
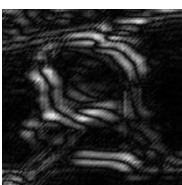
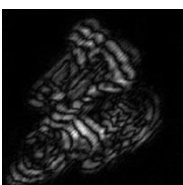



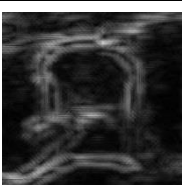
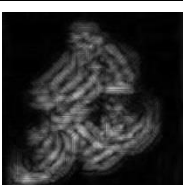


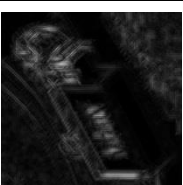
	Person	Cyclist	Car	Bus	Truck
Original Image					
Gabor Filter $\theta=0$					
Gabor Filter $\theta=\pi/4$					
Gabor Filter $\theta=\pi/2$					
Gabor Filter $\theta=3\pi/4$					
Sum of Filter Outputs ($\div 4$)					

Table 2 – Examples of Gabor Filtering output on image samples from the 5 classes using multiplicative convolution

Another key parameter in the HMAX IPP is the size of the templates to extract for matching in the S2 layer. For this part, I ran an experiment with the IPP to test the template size versus the classification percentage. Figure 13 shows the results from using templates of size 4x4 through 16x16. In Mutch and Lowe's work, they consider only templates of size 4x4, 8x8, and 16x16. Note that the template sizes are actually $N \times N \times 4$, which follows from the fact that there are 4 edge orientation channels which were output from the S1 layer. The number of edge orientation channels was not explored in this work.

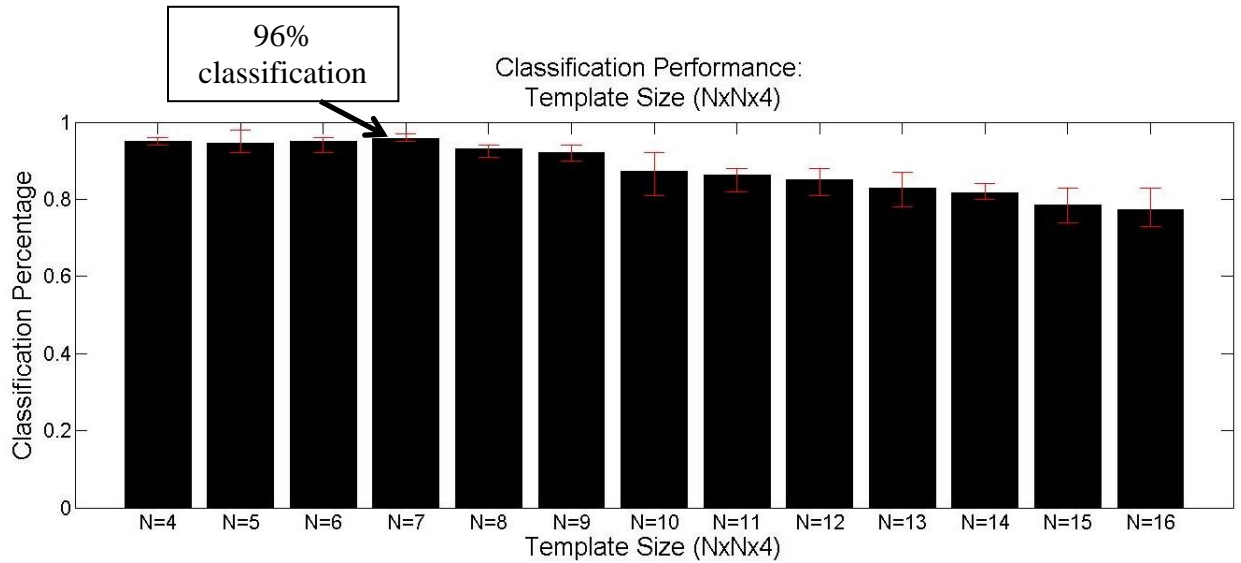


Figure 13 – classification percentages versus S2 template size for the benchmark HMAX IPP

This graph shows the average classification percentage for every template size over 10 runs, with the error bars representing the min and max performances over the 10 runs. According to this data with limited runs, the optimal template size is somewhere between 4x4x4 and 7x7x4. For the benchmark design, I choose to have a template size of 7x7x4, since this is the peak in Figure 13. Another parameter of the S2 layer affects the IPP during the training phase, which is the parameter to determine how many templates to extract from each test image

during training. For the benchmark and the modified IPP, I choose to extract 6 templates per training image, for a total 6 templates x 105 training images = 630 templates of size 7x7x4.

The only parameter to consider in the classification layer is whether or not to normalize the feature vector data, such that the feature dimensions of training data feature vectors all have a cumulative average of 0, and a variance of 1. According to Mutch and Lowe, this improves performance. For this work I also normalize the data. My code uses the autoscale feature of the Matlab SVM classifier to perform this function.

Using the parameters chosen above, the benchmark IPP results that I achieve are summarized in the following table. More specifically, I have chosen to have 2 image scales output from the IL, use a Gabor Filter of size 11x11 in the S1 layer, a template size of 7x7x4 in the S2 layer, and to scale the feature vectors before I train and test the SVM classifier. Table 3 summarizes the information for the benchmark IPP. The classification percentage given in the table is the average performance over 10 runs of the IPP. The differences between runs are the templates that are extracted from the training images, since templates are chosen randomly during the training phase.

Benchmark HMAX based IPP							
# IL Image Scales	S1 Gabor Filter Size	Convolution Type	S2 γ for Template Match	S2 Template Size	Classifier Method	Scaling Feature Vector	Classification Percentage
2	11x11	Multiplicative	2	7x7x4	SVM	Matlab autoscale	<u>0.958</u>

Table 3 – Benchmark IPP Parameters and Classification Percentage Performance

4.3 USING OSCILLATOR MODELS

With the benchmark of the IPP set, I now explore places to replace the parameterized operations given in Table 3 with parameterized operations that oscillators can perform. For my oscillator models, I use both DoM, which is the model I call the oscillator behavior given in Equation 4, and a GRBF model, which is given in Equation 7 in section **3.0 Using GRBF as an Oscillator DoM Model**. The places that I replace oscillator models are:

1. changing the type of computation performed in the convolution in the S1 layer from multiplication to DoM and GRBF,
2. changing the S2 template match computation from GRBF to DoM as well, and
3. modifying the Classifier method from SVM to a Nearest Neighbor approach with GRBF and DoM as the distance metric.

Within each of these modifications, there are many parameters to adjust in order to maximize the performance of the IPP given the modifications. Each of the modifications for the layers are optimized separately, with the rest of the IPP held to the benchmark discussed in the previous section. Then, after an optimization has been determined for each layer, I combine the optimizations and discuss the performance of the modified IPP as a whole, using oscillator models as the computational backbone at the 3 locations discussed. The S1 layer modifications are discussed and optimized in section 4.4, the S2 layer modifications and optimizations are discussed and optimized in section 4.5, and the classifier optimization is discussed in section 4.6. Lastly, section 4.7 discusses the results that I achieve with the complete replacement of all of the sections of the IPP with oscillator models.

4.4 S1 LAYER MODIFICATION

In this section, I explore replacing the dot product using multiplication in the S1 layer with using both a DoM and a GRBF. The idea is that multiplications are computationally expensive, meaning the circuits for multiplications take a long time to compute, and are power hungry. By replacing the multiplications with an oscillator coupling circuit, we should be able to achieve both speedup and power savings.

There are two parameters to tune in the S1 layer. The size of the Gabor Filter and the γ used for the DoM and GRBF oscillator models both impact the performance of the edge detection. Figure 14 shows a sweep of the two variables for DoM and Figure 15 shows a sweep of the two variables for a GRBF response computation. The sweeps are for filter sizes of 5x5 through 11x11, and γ values on a logarithmic scale between 0.001 and 1. Figure 16 shows a sweep of another range of the two variables to demonstrate a better classification performance for GRBF, since the first sweep produced poor performance for a GRBF response. Each run in the figure represents a complete training and testing of an IPP.

Final Classification Performance using DoM in S1: Gamma for filter responses in S1 Layer vs. Filter size (NxN)

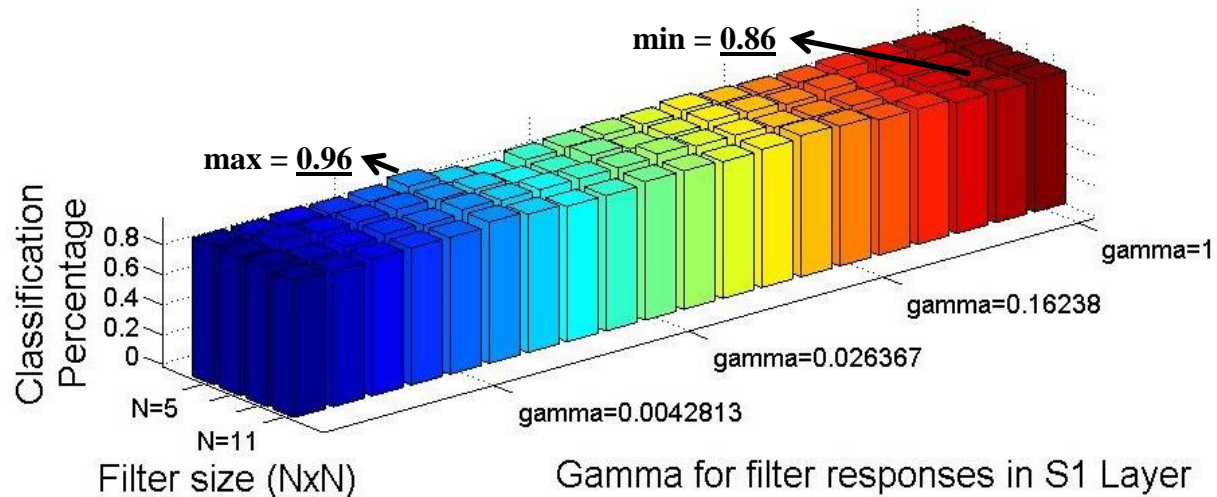


Figure 14 – Sweep of variables γ and Gabor filter size using DoM oscillator model as a replacement for dot product in the convolution in the S1 layer to optimize classification percentage

Final Classification Performance using GRBF in S1: Gamma for filter responses in S1 Layer vs. Filter size (NxN)

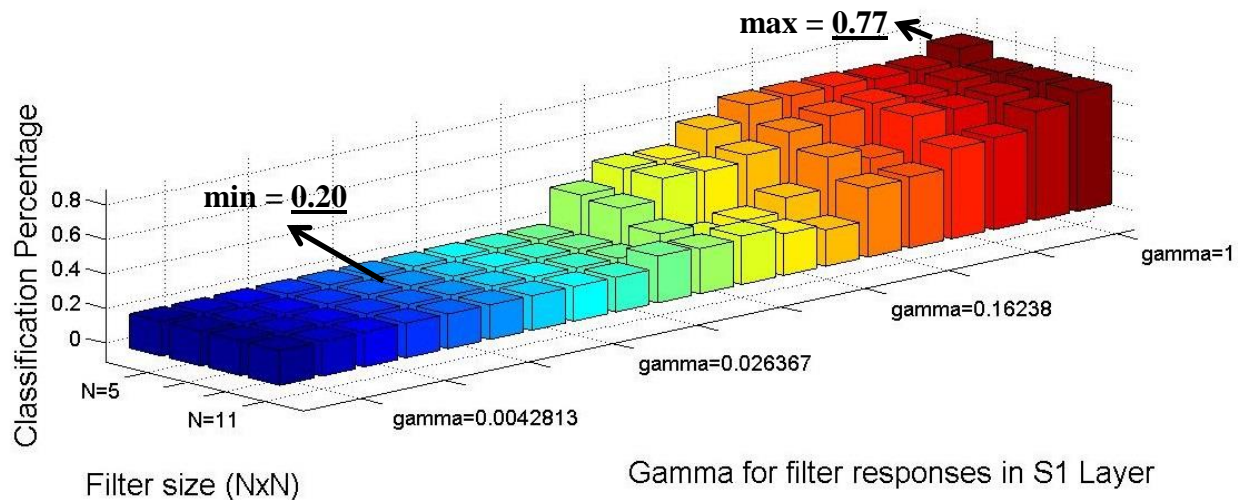


Figure 15 – Sweep of variables γ and Gabor filter size using GRBF oscillator model as a replacement for dot product in the convolution in the S1 layer to optimize classification percentage

Figure 14 demonstrates that the DoM model for oscillators can produce very good results when used in place of the dot product in the S1 layer convolution. There is a wide range of γ values, which produce fairly high results, and in some cases it performs just as well as the benchmark IPP. Figure 15 shows that γ does not scale the same way between DoM and GRBF and that classification percentage on the same range of γ for DoM and GRBF are not comparable. In order to prove that GRBF can also perform well, I ran another sweep of different filter sizes and γ values to produce respectable results for using a GRBF computation as the convolutional backbone. The results of this sweep can be seen in Figure 16. Showing that either model is capable of performing well, while not critical for this work, is a sign that multiple different oscillator behaviors can be used in an IPP to produce good results.

Final Classification Performance using GRBF in S1: Gamma for filter responses in S1 Layer vs. Filter size (NxN)

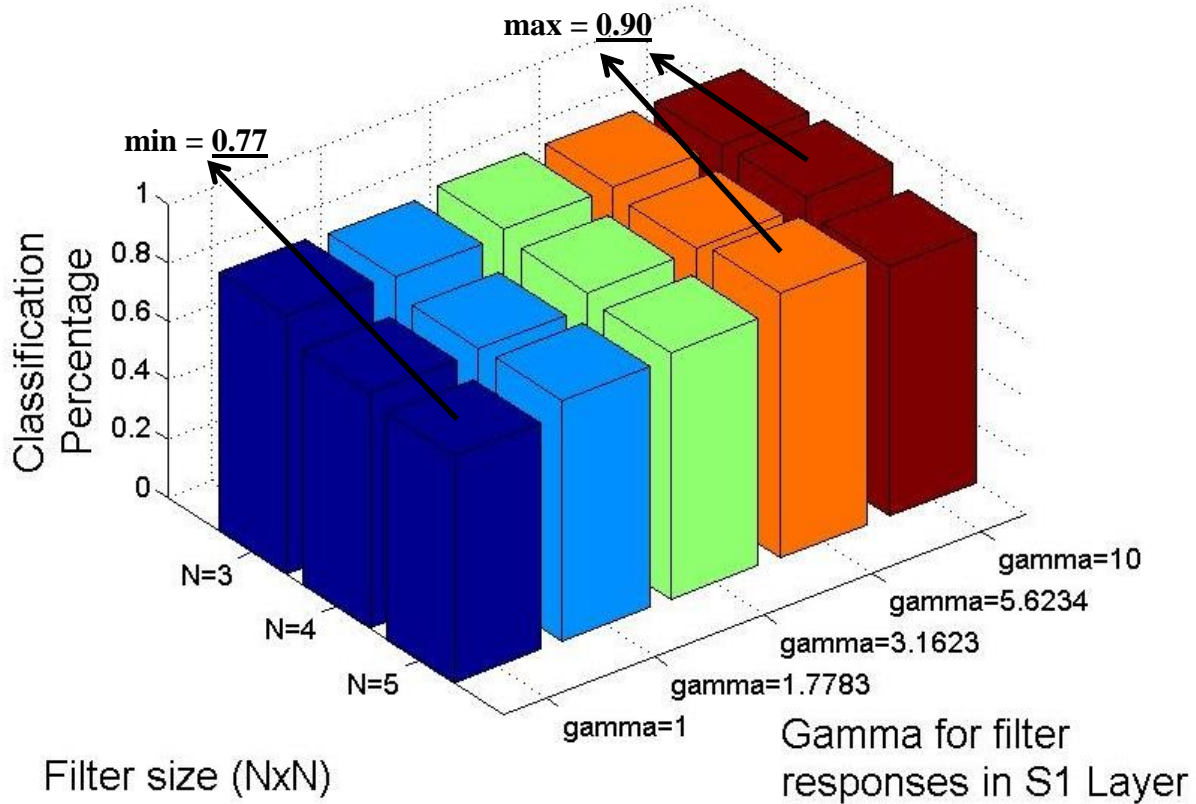


Figure 16 – Second sweep of variables γ and Gabor filter size using GRBF oscillator model as a replacement for dot product in the convolution in the S1 layer to optimize classification percentage

I found that the γ , filter size, and oscillator model that produced the highest classification percent (in Figure 14) was with a classification percentage of 0.96 where $\gamma=0.006$, filter size = 5x5, and the oscillator model was DoM. However, DoM did not see a large change in results depending on the filter size. Therefore, for the modified S1 layer, I choose to use values of $\gamma=0.006$, filter size = 5x5, and the oscillator model as DoM for the optimal performance in the S1 layer.

As a point of comparison between this model and the benchmark, the following table is the result of the same experiment using the optimal parameters that I found that was shown in Table 2 using multiplicative convolution.







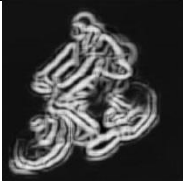


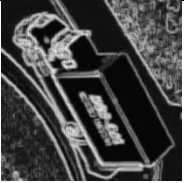
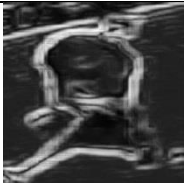
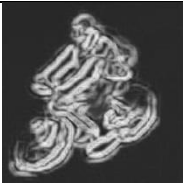


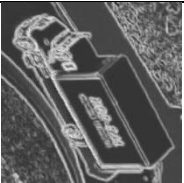
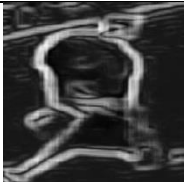
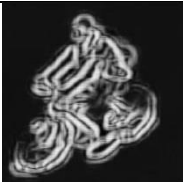


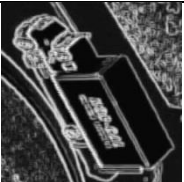
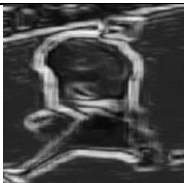
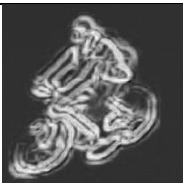


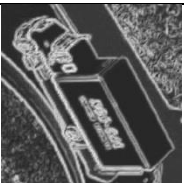

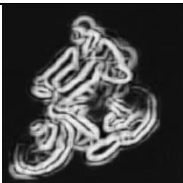


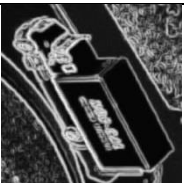
	Person	Cyclist	Car	Bus	Truck
Original Image					
Gabor Filter $\theta=0$					
Gabor Filter $\theta=\pi/4$					
Gabor Filter $\theta=\pi/2$					
Gabor Filter $\theta=3\pi/4$					
Sum of Filter Outputs ($\div 4$)					

Table 4 – Examples of Gabor Filtering output on samples images from the 5 classes using DoM convolution

Looking at Table 2 and Table 4 we can see several advantages and disadvantages to using the DoM filtering computation instead of the multiplication-based dot product. The main advantage is that there are well defined edges in each of the filtering orientations, and there seems to be less noise on the edges. However, the difference between the different orientation channels when using the DoM based convolution is less than in the dot product based convolution, and therefore we lose some information about the orientation of the edges for DoM convolution. Overall, the HMAX IPP performance classifies with about the same accuracy using the two methods, as shown by the maximum classification percentage in Figure 14.

4.5 S2 LAYER MODIFICATION

For the S2 layer, I concluded in a previous section that using templates of $7 \times 7 \times 4$ was the optimum template size for this dataset. Because the template size and the choice of the model used for template matching are independent of one another in terms of their effect on the classification percentage, I also use $7 \times 7 \times 4$ template sizes for the modified IPP.

Next, I determined the optimal γ value for the DoM and GRBF oscillator models when used as template matching distance metrics. In the original work by Mutch and Lowe, they use a value of 2 for γ . For this reason, for both tests I decided to sweep γ equidistant from 2 on both the larger and smaller side of γ on a logarithmic scale. This gives a range of $\gamma=0.25$ to $\gamma=16$ using 7 data points. Figure 17 shows the results from sweeping γ for GRBF over this range and Figure 18 shows the results from sweeping γ for DoM over this range.

The figures show that γ has little impact on the performance of the IPP in this stage for either DoM or GRBF as the template response function. The performance plateaus after γ reaches a certain magnitude, which corresponds to a closer-to linear region in the GRBF and DoM models. However, the larger the γ , the more precision is necessary in order for the models to behave correctly in hardware. This is due to the fact that a larger γ corresponds to a smaller difference between outputs of each of the GRBF and DoM functions for a given difference between input vectors. The tradeoff is to have γ be as small as possible and still have the best possible classification percentage.

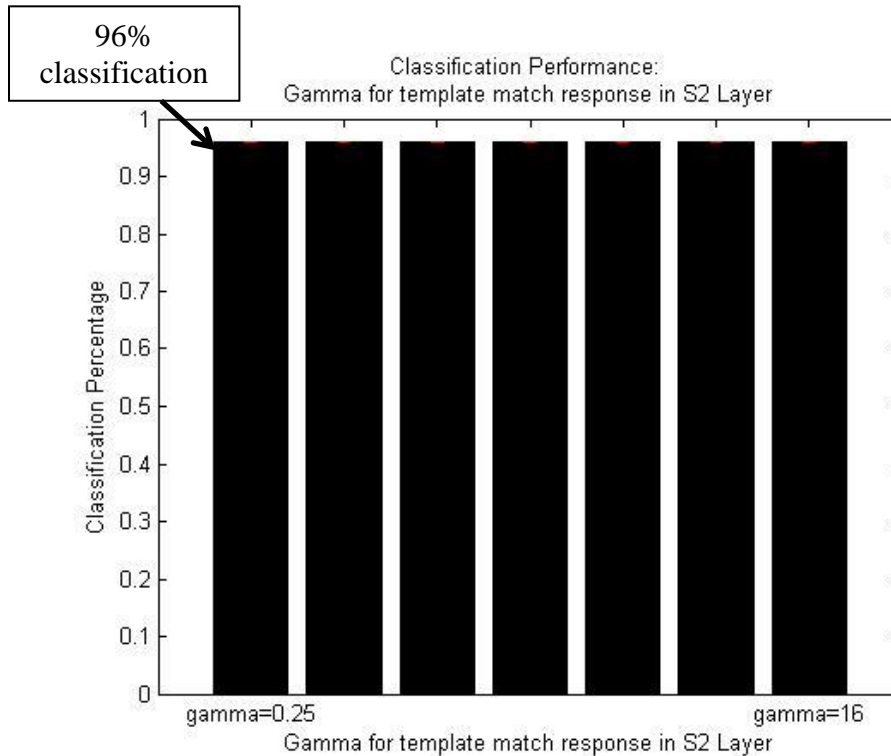


Figure 17 – S2 Layer DoM template response, γ parameter's impact on the classification performance

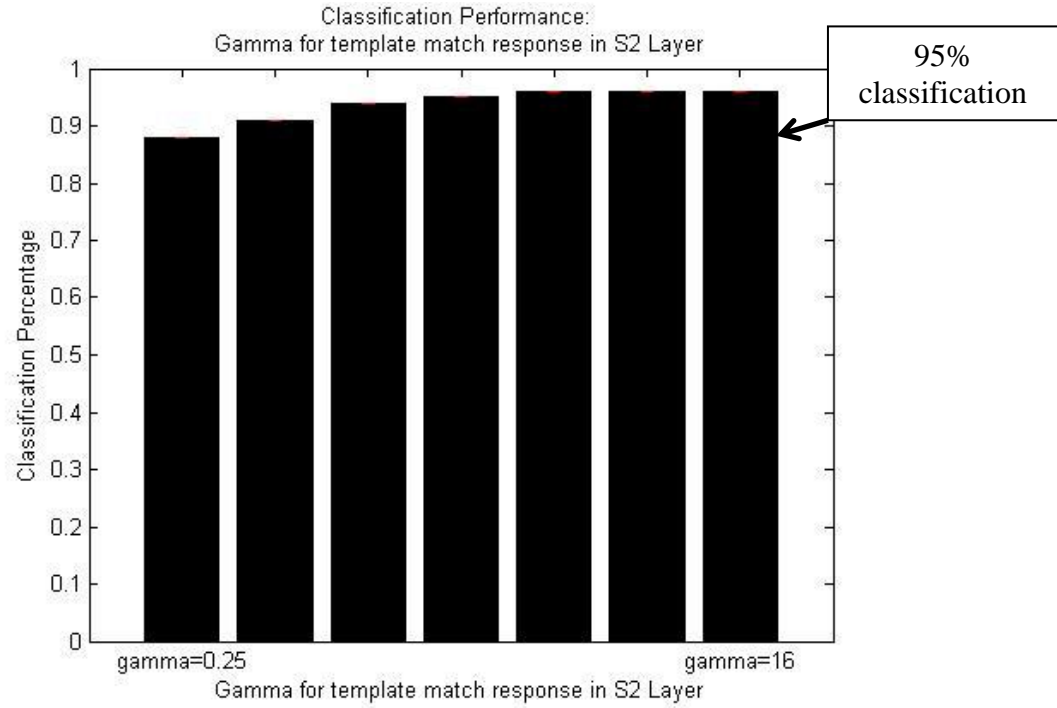


Figure 18 – S2 Layer GRBF template response, γ parameter's impact on the classification performance

Based upon the results shown in Figure 17 and Figure 18 I decided to use a DoM function with a γ of 0.25. The reason that I choose DoM is for consistency, since I also chose DoM for the oscillator model in the S1 layer. Because GRBF and DoM produced very similar results, GRBF could also be used, but for my final design I use DoM as the oscillator model in the S2 layer.

4.6 CLASSIFIER MODIFICATION

For the classifier, I decided that using a nearest neighbor classifier with oscillator models as the distance metric was a good substitute for the SVM considering that I want to use oscillator models as much as possible for the computational workhorse of the classifier. For the

optimization of this layer, I swept γ over the same range as in the S2 layer. Figure 19 shows the results from sweeping γ for GRBF over this range and Figure 20 shows the results from sweeping γ for DoM over this range.

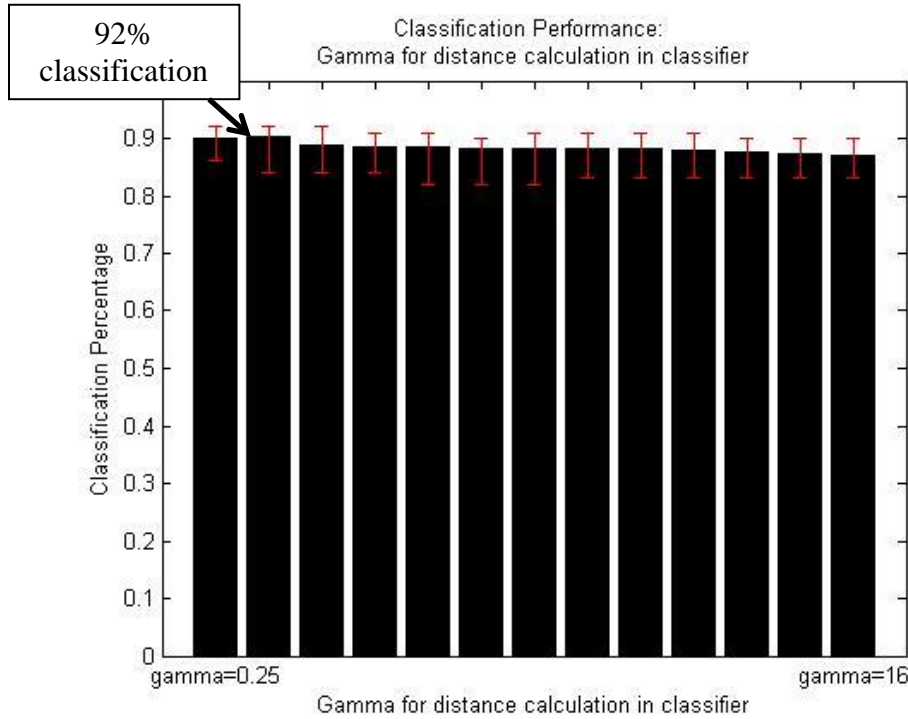


Figure 19 – Classifier DoM template response, γ parameter's impact on the classification performance

With these tests, it is clear that the nearest neighbor classification does not perform as well as an SVM, but still does not perform badly. The maximum classification rate for the nearest neighbor classification was 92%. For the oscillator model that I choose to implement, I choose one of the better classification percentage results, which is to use DoM as the model with $\gamma=0.25$. This is also a choice in order to maximize consistency between the oscillator models used in the IPP, since the S1 and S2 layers use DoM as well for the modified version. Further, in the S2 layer, the same γ value of 0.25 was chosen as well.

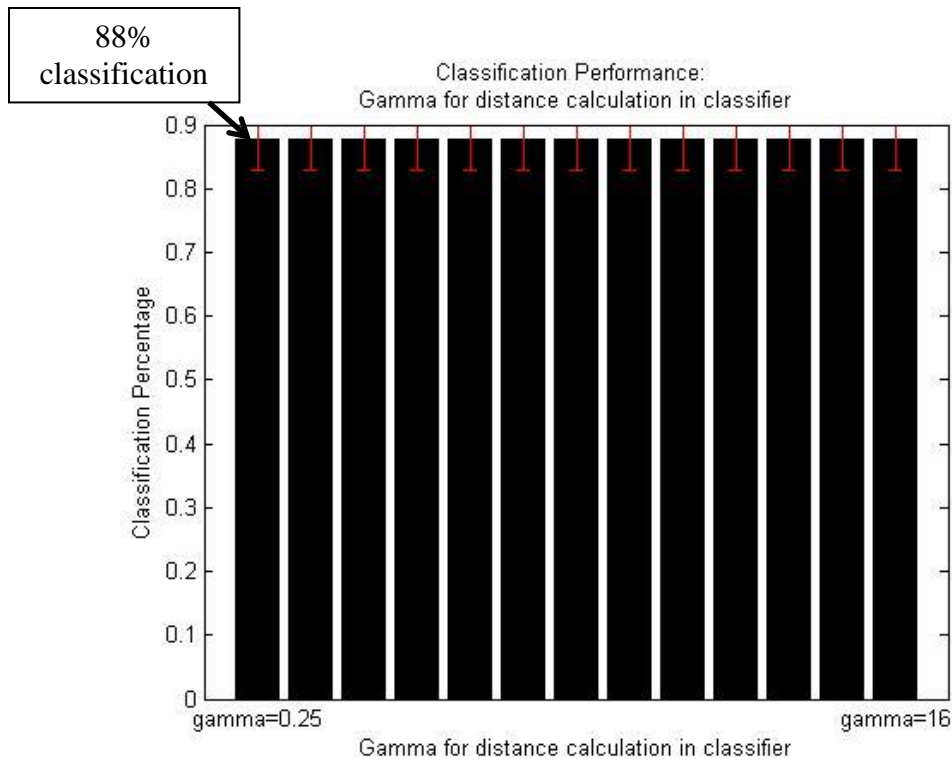


Figure 20– Classifier GRBF template response, γ parameter’s impact on the classification performance

4.7 OVERALL PERFORMANCE AND RESULTS

The following table gives a summary of the DoM oscillator model-based HMAX IPP, along with the data from Table 3 from section 4.2.1. A * in the table denotes a modification from the benchmark IPP. The dataset for both tests was to use 21 training and 20 test images for each of the 5 classes. The results are based upon an average percentage over 10 runs for each case.

Modified HMAX based IPP							
# IL Image Scales	S1 Gabor Filter Size	Convolution Type	S2 γ for Template Match	S2 Template Size	Classifier Method	Scaling Feature Vector	Classification Percentage
2	5x5 *	DoM, $\gamma=0.006$ *	DoM, $\gamma=0.25$ *	7x7x4	DoM nearest neighbor, $\gamma=0.25$ *	none *	<u>0.925</u>
Benchmark HMAX based IPP							
# IL Image Scales	S1 Gabor Filter Size	Convolution Type	S2 γ for Template Match	S2 Template Size	Classifier Method	Scaling Feature Vector	Classification Percentage
2	11x11	Multiplicative	GRBF, $\gamma=2$	7x7x4	SVM	Matlab autoscale	<u>0.958</u>

Table 5 – Summary of oscillator model and parameter modifications to the HMAX IPP, along with the results from the benchmark IPP. A * denotes a change in the parameter from the benchmark IPP

This table shows that a classification drop was experienced from when we use oscillator models as compared to the conventional benchmark IPP algorithm, from an average classification percentage of 0.958 to an average classification percentage of 0.925. In the next chapter, I discuss the parameterization of speedup and power savings based upon the IPP discussed in Table 5. The speedup and lower power consumption justifies using a new hardware approach, since there is only a slight classification hit from the benchmark IPP to the modified IPP where all 3 modifications are considered.

5.0 HMAX MODIFICATION DISCUSSION

In this chapter, I show the potential speedup and power savings achievable by using oscillators as replacements for the computations in the IPP that were discussed in the previous chapter. I show this in multiple steps. First, I parameterize the reduction of computation for both the traditional IPP computation and for the modified IPP's oscillator coupling based computation. Based upon this parameterization, I explore potential speedups depending on the characterization of the oscillator coupling circuit. Secondly, I cite some previous work done to create coupled oscillator systems and show the potential for extremely lower power consumption.

5.1.1 Parameterization of Speedup

In this section, I determine the number and type of operations that are required in each stage of the benchmark IPP, and also the number and type of operations that are performed in each stage of the modified IPP. Using these values, I determine what I call the computational load, which I define as the amount of time to perform all of the necessary operations sequentially, for both the benchmark IPP and the modified IPP. Based upon these values, I compute a computational reduction from using the modified IPP as opposed to the benchmark IPP.

It is important to note the significance of computing the computational reduction, as it is not an implementation specific metric. Regardless of the parallelization of the algorithm, the

computational reduction is a constant. This is an accurate measure of how expensive the algorithm is to run. If we have more hardware, then we can perform more in parallel than if we only had a single computational unit but the total computational load is identical in either case. Further, the data flow and the data dependencies of the benchmark and modified IPPs are identical; therefore the potential for parallelization for both algorithms is similar. For these reasons, the computational reduction is an estimate for the speedup from using the modified IPP as opposed to the benchmark IPP assuming similar parallelization for the benchmark and modified IPPs.

For the S1 layer, I consider the number of dot products that are computed, and the size of each dot product for the benchmark IPP. In the S2 layer, GRBF computations are necessary in the benchmark IPP, so I determine how many multiplications it takes for the distance computation, how many square roots are needed, and how many exponential computations are used. For the classifier, the SVM requires a dot product against each of the support vectors, as well as a summation and another multiplication [38]. I only consider the dot products however, because this is the most computationally expensive part of this step. For each layer in the modified IPP, I consider how many degrees of match are necessary, and the size of each degree of match.

The following table summarizes the number of computations that are necessary for computing each layer in the HMAX IPP for both the benchmark and modified versions. Based on these numbers, I create a performance model, in order to determine a computational reduction equation based upon how long multiplications, exponential calculations, square roots, and DoM calculations take. These results use the parameters from the previous chapters for the respective IPPs, and are based upon the work needed in order to classify a single input image.

	Benchmark IPP	Modified IPP
S1 Layer	dot products: 22504 ⁽¹⁾ filter size: 11x11 = 121 ⁽²⁾ # of multiplications = 2,722,984 ⁽³⁾	DoM calculations: 25120 ⁽¹⁰⁾ filter size: 5x5 = 25 ⁽¹¹⁾
S2 Layer	GRBF calculations: 161,280 ⁽⁴⁾ multiplications: 31,610,880 ⁽⁵⁾ square roots: 161,280 ⁽⁶⁾	DoM calculations: 204,120 ⁽¹²⁾ template size: 7x7x4 = 196 ⁽¹³⁾
Classifier	dot products: 10 ⁽⁷⁾ feature vector size: 630 ⁽⁸⁾ # of multiplications: 6,300 ⁽⁹⁾	DoM calculations: 105 ⁽¹⁴⁾ feature vector size: 630 ⁽¹⁵⁾

Table 6 – Number of computations which are the main computational differences between the benchmark and modified IPP. These are used as the basis for determining the computational reduction of the modified IPP. The superscripts are pointers to the list below to show how to compute the value shown in the table.

- (1) 2 sizes of input images: 112x112 and 120x120 for a total of 102x102 + 110x110 dot products (with 11x11 filters) = 22,504 dot products
- (2) Each dot product is of size 11x11 = 121
- (3) # of multiplications = # of dot products x size of each dot product = 22,504 * 121 = 2,722,984 multiplications
- (4) By GRBF, I mean solely the exponential calculation. There are 630 templates extracted. S2 layer input is of size 22x22x4. Templates are of size 7x7x4, for a total of 16x16 = 256 template matches per template. 256*630 templates = 161,280 template matches.
- (5) # of distance computations = # GRBFs = 161,280. # of multiplications per distance computation =
- (6) # square roots needed = # distance computations that are performed = # GRBFs performed = 161,280.
- (7) # of dot products = # of SVMs needed for the one-to-one multiclass SVM classifier. 5 classes necessitates 4+3+2+1 = 10 SVMs, for 10 total dot products.
- (8) The feature vector size = # templates extracted = 630.
- (9) # of multiplications = # of dot products x size of dot product = 630 x 10 = 6,300.
- (10) 2 sizes of input images: 112x112 and 120x120 for a total of 108x108 + 116x116 = 25,120 DoMs
- (11) Each DoM has vector inputs of size 5x5 = 25

(12) There are 630 templates extracted. S2 layer input is of size $24 \times 24 \times 4$. Templates are of size $7 \times 7 \times 4$, for a total of $18 \times 18 = 324$ template matches per template. 324×630 templates = 161,280 template matches.

(13) # template match vector size = $7 \times 7 \times 4 = 196$

(14) # DoM computations = # of training images = 105

(15) The feature vector size = # templates extracted = 630.

Here, I define the parameters for my computational reduction equation for the modified IPP versus the benchmark IPP. For this equation, I only consider the computational reduction of these three layers, and I do not consider the IL, C1 nor C2 layers. Based upon my software simulations in Matlab, the S1 and S2 layers accounted for 93% of the total feed forward run time of classification for the benchmark IPP.

For the purposes of defining computational load performed by the benchmark IPP, I consider the term M, which I call the “time to perform a multiplication”. This is a general term, since there are other computations besides multiplications in Table 6. To account for this, I find the time necessary to perform square roots and GRBF in terms of M based upon simulation results in Matlab and a compiled C++ program. This way, I have a simple way of parameterizing the computational load of the benchmark IPP. Even though an increase in the time for multiplications does not necessarily correlate to increase the time for square roots, multiplications are the dominant computations necessary for the benchmark IPP so parameterizing the computational load based upon M is a good estimate.

For this work, I determine that the time for a GRBF = $\sim 123M$ and time to compute a square root = $\sim 32M$. This is based upon a simple test run in the Matlab environment to compare their speed of execution. Note that by GRBF, I mean only the time to compute the exponential, since the computational load of finding the distance is based upon the size of the input vectors. I

consider the time it takes to compute the distance separately and base the distance computation in terms of the number of necessary multiplications and square roots. Based upon this assumption, the following equation gives the total computational load (CL) in terms of M for the benchmark IPP. This can be calculated as a result of the values in Table 6, in combination with the assumption that a GRBF = 123M, and sqrt = 32M. I compute the computational load of each of the layers, and then compute the total computational load in order to modularize the analysis. For notational purposes, *bm* is an abbreviation for benchmark, and *mod* is an abbreviation for modified, with respect to which IPP I am referring.

$$\begin{aligned}
CL_{bm,S1} &= 2,722,984M \\
CL_{bm,S2} &= 161,280 \times GRBF + 31,610,880 \times M + 161,280 \times sqrt = 56,609,280M \\
CL_{bm,classifier} &= 6,300M \\
CL_{bm} &= (CL_{bm,S1} + CL_{bm,S2} + CL_{bm,classifier}) \\
\Rightarrow CL_{bm} &= (2,722,984 + 56,609,280 + 6,300)M = \underline{59,338,564M}
\end{aligned}$$

Equation 8 – Formulas for determining the computational load of the benchmark IPP

In order to define the computational workload of the modified IPP, I define two parameters: the length of time necessary to perform a degree of match (D), and the size of the vector used in computing a degree of match using oscillators (k). k is an important parameter since in actual hardware, arbitrarily large input vectors run into physical limitations for coupling from the oscillators themselves. The amount of work for each layer using this model is the number of DoMs, multiplied by the ceiling of the vector size divided by k. This is based on the assumption that a DoM model for larger input vectors can be comprised as a sum of results from smaller DoM calculations from input vector sizes. I compute the computational load of each of the layers of the modified IPP, and then compute the total computational load in order to modularize the analysis.

$$CL_{\text{mod},S1} = 25,120 \times D \times \left\lceil \frac{25}{k} \right\rceil$$

$$CL_{\text{mod},S1} = 204,120 \times D \times \left\lceil \frac{196}{k} \right\rceil$$

$$CL_{\text{mod},\text{classifier}} = 105 \times \left\lceil \frac{630}{k} \right\rceil$$

$$CL_{\text{mod}} = CL_{\text{mod},S1} + CL_{\text{mod},S2} + CL_{\text{mod},\text{classifier}}$$

$$\Rightarrow CL_{\text{mod}} = \left(25,120 \times \left\lceil \frac{25}{k} \right\rceil + 204,120 \times \left\lceil \frac{196}{k} \right\rceil + 105 \times \left\lceil \frac{630}{k} \right\rceil \right) D$$

Equation 9 – Formulas for determining the computational load of the modified IPP

Based upon the two computational loads of the modified and benchmark IPPs, I compute the computational reduction as a ratio of workloads. Note that this is a good model for determining the speedup since the computational load is independent of the parallelization. The parallelization is similar for the two IPPs, since the data flow and therefore data dependencies are identical; the difference between the two is the cost of the computations that need to be performed. The following equations give the computational reduction as a result of using the modified IPP instead of the benchmark IPP for each of the layers, and also overall in the IPP architecture.

$$\begin{aligned}
CR_{s1} &= \frac{2,722,984}{25,120 \times \left\lceil \frac{25}{k} \right\rceil} \times \frac{M}{D} \\
CR_{s2} &= \frac{56,609,280}{204,120 \times \left\lceil \frac{196}{k} \right\rceil} \times \frac{M}{D} \\
CR_{classifier} &= \frac{6,300}{105 \times \left\lceil \frac{630}{k} \right\rceil} \times \frac{M}{D} \\
CR &= \frac{59,338,564}{\left(25,120 \times \left\lceil \frac{25}{k} \right\rceil + 204,120 \times \left\lceil \frac{196}{k} \right\rceil + 105 \times \left\lceil \frac{630}{k} \right\rceil \right)} \times \frac{M}{D}
\end{aligned}$$

Equation 10 – Computational reduction (CR) for each layer, and for overall IPP as a function of the time to multiply using CMOS (M), the time for a DoM to complete using oscillators (D), and the number of coupled oscillators in the oscillator circuit (k)

I now consider two parameters for the computational reduction equation as the ratio of the time to compute a degree of match versus time to compute a multiply (D/M), and the coupling capacity of the oscillator circuit (k), and their effect on the computational reduction.

To give an estimation of the value for D/M, I estimate the amount of time it takes for convergence to happen on STO oscillators, and I estimate the time it takes for an Intel processor to execute a 64 bit floating point multiplication. According to [8], synchronization for the STO oscillators take around 6-10 cycles of oscillation, and are able to oscillate “on the order of a few – a few tens of GHz”. This means that we can expect a time to couple of roughly 1-10 nanoseconds, with slightly longer or shorter times also possible. Estimating the amount of time necessary to compute a 64-bit multiplication on a pipelined processor is complicated because the latency is not a good estimate of serialized code. To give a range of values possible depending on the program’s implementation, a multiplication will take between 1 cycle and the number of

stages in a pipeline. In past years, Intel processor pipelines have had depths of 20 stages, making multiplications take between 0.3ns-6ns, depending on the number of multiplies that need to be performed, assuming a clock cycle of ~3GHz [49]. For this reason, I decided to sweep the D/M parameter from 0.25 to 32 to account for potential conditions, and I chose k from 4 to 64, and output the corresponding computational reduction (CR). This plot is shown in Figure 21 for each of the layers as well as the overall IPP.

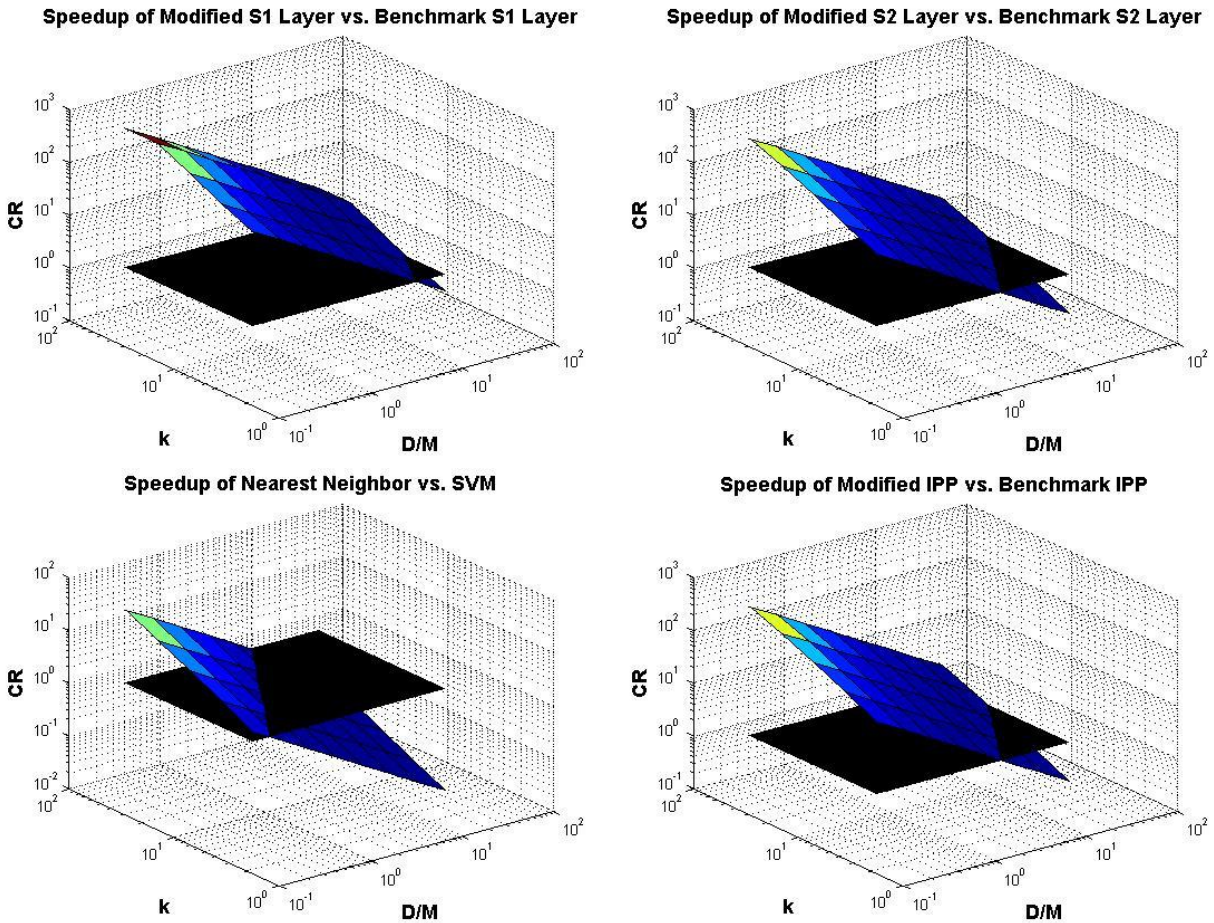


Figure 21 - Plot of the computational reduction of (top left, top right, bottom left, bottom right) S1 layer, S2 layer, classifier, and entire IPP backend as a function of time to compute DoM (D), time to compute a multiplication in CMOS (M), and number of coupled oscillators in the DoM circuit (k). The black plane in each graph represents a computational reduction of 1.

To demonstrate the computational reduction in each of these graphs, I give eight CR values based upon different values for D/M and k in Table 7 from the graphs in Figure 21.

	S1 Layer	S2 Layer	Classifier	Overall
D/M = 0.25, k = 64	434x	277x	24.0x	282x
D/M = 0.25, k = 16	217x	85.3x	6.00x	87.6x
D/M = 1, k = 64	108x	69.3x	6.00x	70.4x
D/M = 1, k = 16	54.2x	21.3x	1.50x	21.9x
D/M = 4, k = 64	27.1x	17.3x	1.50x	17.6x
D/M = 4, k = 16	13.5x	5.33x	0.375x	5.48x
D/M = 16, k = 64	6.77x	4.33x	0.375x	4.40x
D/M = 16, k = 16	3.39x	1.33x	0.0938x	1.37x

Table 7 – CR values for three examples of D/M and k parameter pairings

To extrapolate from the results in Table 7, note that D/M has a linear relationship with CR given a constant k, and k also has approximately a linear relationship given a constant D/M. Table 7 shows the capability of computational reduction between the benchmark IPP and the modified IPP, depending on the parameters D, M, and k. The next table is a summary of the results. The results shown are 1) what percent of the computation is performed in a given layer according to Table 6, 2) what is the computational reduction assuming that we have D/M=1, and k=64, and 3) what was the classification percentage for this configuration. The classification percentages are taken from the results given in Chapter 4.0 HMAX. These characteristics show how effective coupled oscillators are at increasing performance in each of these layers independently of one another.

Name of the Layer	S1 Layer	S2 Layer	Classifier	Overall
Relative % of total computational load for 3 layers in benchmark	4.59%	95.4%	0.0106%	100%
Computational reduction assuming $k=64$, $D/M = 1$	108x	69.3x	6.00x	70.4x
Classification Percentage for Modifying layer(s)	96%	96%	92%	92.5%

Table 8 – Summary of the results in this work. Each column represents a location in the algorithm for a modification to the benchmark IPP, and each row specifies a result associated with that modification.

This table shows that we can achieve a large computational reduction by using coupled oscillators instead of the traditional multiplications, square roots, and exponentials. However, this reduction is largely seen by the S1 and S2 layers, and the classifier does not have the same benefit. However, this is acceptable since the computational load of the SVM in the benchmark IPP is so small compared to the S1 and S2 layers. Further, replacing the SVM with a nearest neighbor classifier constitutes the largest hit in the classification percentage in the modified IPP. If we replace only the S1 and/or S2 layer(s), we observe a comparable classification percentage, while at the same time observing the largest computational reduction.

5.1.2 Discussion of Power Savings

In a progress report from Intel, there are results which show a large reduction in power for the use of coupled oscillators to perform Gabor Filtering over an input matrix [8]. Of the oscillator technologies discussed, they research the use of Spin-Torque Oscillators (STOs) and Resonant Body Oscillators (RBOs). These oscillator technologies have been studied as a possible backbone for an associative memory architecture [50] [51] [52].

Based on the work done in [8], there is evidence to support that oscillator arrays will be able to perform 4 orders of magnitude more efficiently than a general purpose processor, and up

to 2 orders of magnitude more efficiently than an ASIC accelerator in terms of energy consumption. Further, the work done in [50], [51], and [52] shows further evidence that STOs and RBOs show promise in being able to perform the computations necessary for this work.

Here, I perform some analysis using some assumptions to show the power savings that oscillators are hypothesized to offer. According to [53] and [54], STO oscillators can operate at about 0.1-10mA, with an operating resistance ranging between 1-500 Ω and an oscillation waveform of 1-15GHz. Assuming a current of 1mA, resistance of 10 Ω , frequency of 10GHz, and 10 cycles of oscillation before coupling, then this corresponds to 10fJ/bit for oscillator coupling. According to [55], a low power 16-bit multiplier consumes roughly 540 μ W. These multiplications are performed at 1GHz and thus this corresponds to an energy consumption of 540 μ J, or 34 μ J/bit. This brief analysis demonstrates the several orders of magnitude power savings that other researchers have claimed are achievable using these oscillators [8] [50] [52].

6.0 CONCLUSION, SUMMARY, AND FUTURE WORK

6.1 SUMMARY

Using an approach based on the use of coupled oscillators as a basic computational primitive, this work used the benefits gained from using this new computational paradigm to gain performance in terms of both speed and power with respect to IPPs, without decreasing the accuracy of their algorithms. More specifically, this work explored the use of coupled oscillators to perform non-Boolean computation. By non-Boolean computation, for the purposes of discussion in this thesis, I mean a set of computations that are not based on Boolean operators OR, AND, etc. My work provides empirical evidence that inserting models which represent the behavior of coupled oscillators into an existing HMAX IPP algorithm does not compromise the accuracy of the image recognition, while at the same time provides a methodology for future work of inserting oscillators into an HMAX IPP hardware implementation in order to create faster, more efficient architectures. The work that I do to prove this is to construct a benchmark IPP based on the HMAX design, and also construct a modified version of the algorithm, which uses coupled oscillator models as the primary computation. I compare the accuracy of the classification percentages between the two implementations to show the algorithmic effectiveness of the oscillator. Finally, I justify the oscillator-based implementation based upon

research evidence that using coupled oscillators improves performance of calculation over CMOS implementations in both speed and power.

6.2 CONCLUSIONS

My conclusions from this work are three-fold. First, I conclude that oscillator models are definitely able to perform computations within the HMAX IPP structure without compromising the accuracy of the algorithm. I show this result in chapter 4.0 HMAX, and summarize the results in Table 5. Second, I conclude that oscillators effectively reduce the amount of work necessary compared to typical HMAX implementation, since they are able to replace many of the expensive operations within the IPP. This result was shown in 5.0 HMAX Modification Discussion. Last, based upon current research trends, this speedup can be achieved while also decreasing the amount of power that is necessary for the computations. This final result is seen in Section 5.1.2 Discussion of Power Savings.

For this work, I proposed three locations within the HMAX IPP of inserting oscillator models: in the S1 layer, in the S2 layer, and also to replace the SVM classifier. Using a DoM operation instead of a dot product as the fundamental computation performed in the convolution in the S1 layer, I was able to achieve the same accuracy as the benchmark implementation. Further, with certain assumptions about the speed of the oscillators, I was able to achieve a computational reduction of two orders of magnitude over the benchmark architecture. In the S2 layer, I was able to use the DoM model to replace the template response function in the benchmark implementation to achieve identical classification percentages as well. Further, using the DoM model to replace the GRBF template response using CMOS based computation I was

able to save over three orders of magnitude in computational load. Finally, replacing the SVM had less successful results, with only a maximum of 6X speedup while also reducing the accuracy of the IPP from a 95.8% classification percentage to a 92% classification percentage. Based upon these results, I conclude that using the oscillator models in only the S1 and S2 layers will maximize the performance increase while not compromising the accuracy of the IPP.

6.3 FUTURE WORK

Based on my observations and conclusions, there are many future research projects to explore. First, this work strongly motivates the future research of using coupled oscillators to perform pattern matching, and more general applications of high dimensional distance metrics. Second, this work proves that using coupled oscillators as a computational primitive is effective in their use in the HMAX IPP, and constructing a hardware IPP with this design would increase performance over other accelerator implementations. Last, future work should explore using oscillator models in other IPP architectures discussed in **2.2 Previous Work on image processing**. Convolution is a typical computation performed in several IPPs, including CNNs. Because HMAX successfully uses a DoM as a replacement for the dot product in the convolution in the S1 layer, there should be a method to implement this modification in CNNs, and other IPPs as well.

7.0 BIBLIOGRAPHY

- [1] F. Faggin, M. Hoff, S. Mazor and M. Shima, "The history of the 4004," *Micro, IEEE*, vol. 16, no. 6, pp. 10,20, 1996.

- [2] E. Karl, Y. Wang, Y.-G. Ng, Z. Guo, F. Hamzaoglu, M. Meterelliyo, J. Keane, U. Bhattacharya, K. M. K. Zhang and M. Bohr, "A 4.6 GHz 162 Mb SRAM Design in 22 nm Tri-Gate CMOS Technology With Integrated Read and Write Assist Circuitry," *Solid-State Circuits, IEEE Journal of*, vol. 48, no. 1, pp. 150,158, 2013.

- [3] Satish Damaraju, Varghese George, Sanjeev Jahagirdar, Tanveer Khondker, R. Milstrey, Sanjib Sarkar, S. Siers, I. Stolerio and A. Subbiah, "A 22nm IA multi-CPU and GPU System-on-Chip," *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pp. 56,57,19-23, 2012.

- [4] T. Mudge, "Power: A first-class architectural design constraint," *Computer*, vol. 34, no. 4, pp. 52-58, 2001.

- [5] G. e. a. Semeraro, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on. IEEE*, 2002.
- [6] N. B. e. a. Rizvandi, "Linear combinations of dvfs-enabled processor frequencies to modify the energy-aware scheduling algorithms," *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, 2010.
- [7] H. Fuketa, M. Nomura, M. Takamiya and T. Sakurai, "Intermittent resonant clocking enabling power reduction at any clock frequency for 0.37V 980kHz near-threshold logic circuits," *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International* , vol. 17, no. 21, pp. 436,437, 2013.
- [8] M. Pufall and e. al., "Feasibility study concerning ultra low power non-boolean systems - annual report," Intel, 2012.
- [9] C. Gnegy and e. al., "Associative Processing using Coupled Oscillators," *Journal of Emerging Technologies in Computing*, 2009 (Submitted).
- [10] T. Shibata and e. al., "CMOS Supporting Circuitries for Nano-Oscillator-Based Associative Memories," *13th International Workshop on Cellular Nanoscale Networks and*

their Applications (CNNA), pp. 1-5, 2012.

- [11] C. Garcia and M. Delakis, "Convolutional face finder: A neural architecture for fast and robust face detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 11, pp. 1408-1423, 2004.
- [12] T. Acharya and A. K. Ray, *Image Processing: Principles and Applications*, John Wiley & Sons, 2005.
- [13] A. e. a. Al Maashri, "A hardware architecture for accelerating neuromorphic vision algorithms," *Signal Processing Systems (SiPS), 2011 IEEE Workshop on. IEEE*, 2011.
- [14] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, pp. 33-61, 1995.
- [15] "Neovision2 dataset - iLab - University of Southern California," University of Southern California, 2013. [Online]. Available: <http://ilab.usc.edu/neo2/dataset/>. [Accessed 12 November 2013].
- [16] T. Kadir and M. Brady, "Saliency, Scale and Image Description," *International Journal of Computer Vision*, vol. 45, no. 2, pp. 83-105, 2001.

- [17] B. A. Olshausen, C. H. Anderson and D. C. Van Essen, "A neurobiological model of visual attention and invariant pattern recognition based on dynamic routing of information," *The Journal of Neuroscience*, vol. 13, no. 11, pp. 4700-4719, 1993.
- [18] J. K. Tsotsos and e. al, "Modeling visual attention via selective tuning," *Artificial intelligence*, vol. 78, no. 1, pp. 507-545, 1985.
- [19] X. Hou and L. Zhang, "Saliency Detection: A Spectral Residual Approach," *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pp. 1-8, 2007.
- [20] L. Itti, C. Kock and E. Niebur, "A Model of Saliency-Based Visual Attention for Rapid Scene Analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 11, pp. 1254-1259, 1998.
- [21] F. Miau and L. Itti, "A Neural Model Combining Attentional Orienting to Object Recognition: Preliminary Explorations on the Interplay Between Where and What," *Engineering in Medicine and Biology Society, 2001. Proceedings of the 23rd Annual International Conference of the IEEE*, vol. 1, 2001.
- [22] H. Uwe and e. al., "An image processing system for driver assistance," *Image and*

Vision Computing, vol. 18, no. 5, pp. 367-376, 2000.

- [23] A. Radhakrishna and S. Susstruck, "Saliency Detection Using Maximum Symmetric Surround," *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pp. 2653-2656, 2010.
- [24] J. Mutch and D. Lowe, "Multiclass Object Recognition with Sparse, Localized Features," *Computer Vision and Pattern Recognition*, vol. 11, no. 18, pp. 17-22, 2006.
- [25] R. Mehrotra, K. R. Namuduri and N. Ranganathan, "Gabor filter-based edge detection," *Pattern Recognition*, vol. 25, no. 12, pp. 1479-1494, 1992.
- [26] J. P. Jones and L. A. Palmer, "An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex," *Journal of Neurophysiology*, vol. 58, no. 6, pp. 1233-1258, 1987.
- [27] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in a cortex," *Nature America Inc.*, vol. 2, no. 11, pp. 1019-1025, 1999.
- [28] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, p. 106, 1962.

- [29] B. Widrow and M. Hoff, "Adaptive switching circuits," *IRE WESCON Convention Record*, pp. 96-104, 1960.
- [30] X. Yao, "Evolutionary Artificial Neural Networks," *International Journal of Neural Systems*, vol. 4, no. 3, pp. 203-222, 1993.
- [31] D. F. Specht, "A general regression neural network," *Neural Networks, IEEE Transactions on*, vol. 2, no. 6, pp. 568, 576, 1991.
- [32] C.-T. Lin and C. S. G. Lee, "Neural-network-based fuzzy logic control and decision system," *Computers, IEEE Transactions on*, vol. 40, no. 12, pp. 1320,1336, 1991.
- [33] H. Rowley, S. Baluja and T. Kanade, "Neural network-based face detection," *Pattern Analysis and Machine Intelligence*, vol. 20, no. 1, pp. 23,38, 1998.
- [34] D. Broomhead and D. Lowe, "Multivariable Functional Interpolation and Adaptive Networks," *Complex Systems* 2, pp. 321-355, 1988.
- [35] J. Park and I. Sandberg, "Universal Approximation Using Radial-Basis-Function Networks," *Neural Computation*, vol. 3, no. 2, pp. 246-257, 1991.

- [36] P. Simard, D. Steinkraus and J. C. Platt, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis," *ICDAR*, vol. 3, 2003.
- [37] C. Nebauer, "Evaluation of convolutional neural networks for visual recognition," *Neural Networks, IEEE Transactions on*, vol. 9, no. 4, pp. 685-696, 1998.
- [38] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *Neural Networks, IEEE Transactions on*, vol. 13, no. 2, pp. 415-425, 2002.
- [39] J. A. Suykens, Vandewalle and Joos, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293-300, 1999.
- [40] C. Farabet and e. al., "Hardware accelerated convolutional neural networks for synthetic vision systems," *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on. IEEE*, 2010.
- [41] Y. C. P. e. a. Cho, "Exploring Gabor filter implementations for visual cortex modeling on FPGA," *Field Programmable Logic and Applications (FPL), 2011 International Conference on. IEEE*, 2011.
- [42] J. e. a. Sabarad, "A reconfigurable accelerator for neuromorphic object

- recognition," *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific. IEEE*, 2012.
- [43] *Horologium oscillatorium*, 1673.
- [44] R. FitzHugh, "Impulses and physiological states in theoretical models of nerve membrane," *Biophysical journal*, vol. 1, no. 6, pp. 445-466, 1961.
- [45] N. Jinichi, S. Arimoto and S. Yoshizawa, "An active pulse transmission line simulating nerve axon," *Proceedings of the IRE*, vol. 50, no. 10, pp. 2061-2070, 1962.
- [46] B. Van Der Pol, "VII. Forced oscillations in a circuit with non-linear resistance. (Reception with reactive triode).," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 3, no. 13, pp. 65-80, 1927.
- [47] B. Van der Pol and J. V. d. Mark, "Frequency demultiplication," *Nature* 120, pp. 363-364, 1927.
- [48] H. Wilson and J. Cowan, "Excitatory and inhibitory interactions in localized population of model neurons," *Biophysical journal*, vol. 12, no. 1, pp. 1-24, 1972.
- [49] G. Hinton and e. al., "The microarchitecture of the Pentium® 4 processor," *Intel*

Technology Journal, 2001.

- [50] C. Gyorgy and e. al., "Spin Torque Oscillator (STO) Models for Applications in Associative Memories," *this conferences*, 2011.
- [51] A. Horvath and e. al., "Synchronization in cellular spin torque oscillator arrays," *Cellular Nanoscale Networks and their applications (CNNA), 13th International Workshop on. IEEE*, 2012.
- [52] D. D. Nikonov and e. al., "Coupled-oscillator associative memory array operation," *arXiv preprint arXiv*, vol. 1304, no. 6125, 2013.
- [53] S. Kaka and e. al., "Mutual phase-locking of microwave spin torque nano-oscillators," *Nature*, vol. 437, no. 15, pp. 389-392, 2005.
- [54] Z. Zeng and e. al., "Ultralow-current-density and bias-field-free spin-transfer nano-oscillator," *Scientific Reports*, vol. 3, no. 1426, pp. 1-5, 2013.
- [55] S. K. Hsu and e. al., "A 110 GOPS/W 16-bit multiplier and reconfigurable PLA loop in 90-nm CMOS," *Solid-State Circuits, IEEE Journal of*, vol. 41, no. 1, pp. 256-264, 2006.

- [56] G. Griffin, A. Holub and P. Perona, "Caltech-256 Object Category Dataset," (Unpublished), 2007.
- [57] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659-1671, 1997.